98を98%使う本

PC-9801 プログラマーズ Bible

東京理科大学EIC／編

技術評論社

98を98%使う本

PC-9801 プログラマーズ Bible

東京理科大学EIC／編

技術評論社

98を98%使う本

PC-9801
プログラマーズ
Bible

東京理科大学EIC／編

技術評論社

■本書のサンプルプログラムはBorland社系のC言語（Turbo C／C++，Borland C++）で書かれております．他の処理系では，システム関係の一部の命令（outportb）などに違いがありますので御注意ください．

■本書掲載のプログラムはディスクサービスを行っています．詳しくは巻末をご覧ください

■本書掲載のプログラムおよび付属ディスクの使用により生じたいかなる損害についても，弊社は責任を負いかねますので，御了承ください．

■本書の内容に関する御質問は，すべて封書でお願い致します．お電話でのお問い合わせには，一切お答えできません．

本書は都合により第1版と内容が一部異なっておりますので,御注意ください.

はじめに

現在，パソコンのプログラミングには2つの大きな方向があります．1つはWINDOWS上のアプリケーションに代表されるような，ハードウェアの違いによらない共通性の高いプログラミング，そしてもう1つは，特定のハードウェアに強く依存する代わりに，そのハードウェアの性能を100％引き出そうとするようなプログラミングです．

これら2つの方向のプログラミングはいずれ劣らず大切なものなのですが，前者のような共通性の高いプログラムを作るための参考文献は数多くあるのに対し，後者のようなハードウェアの性能をフルに引き出すようなプログラムを作るための参考文献は，十分にあるとはいえません．そのうえ，そのようなプログラムを組む場合，有用な情報が多くの文献に断片的に書かれているので，少し本格的なプログラムを作ろうとすると文献を山積みにしてとっかえひっかえ参照しなければならない，ということもざらにあります．

そこで本書では，現在最も普及しているパソコンであるNECのPC-98シリーズをターゲットにして，初心者の方も含めた読者の方々が，PC-98シリーズのハードウェアの性能を100％引き出すようなプログラム（たとえばゲーム，アニメーション，通信ソフトなど）を，本書1冊の参照のみで組めるようにする，ということを目標にしています．そのために本書では，98の性能をフルに引き出すために必要不可欠な，98のハードウェア，I／Oポート，BIOS，LIOなどについて，できるだけわかりやすく，かつ詳細に記述するようにしました．そしてさらに，現実のプログラミング法をより明確に理解していただくために，できるだけ簡潔でわかりやすいサンプルプログラムを多く付属させるようにし，その際の使用言語としては，初心者の方にも理解しやすいように極力C言語を多く使うようにしました．

この本が，読者の方々のプログラミングの友になり得れば望外の幸せです．

なお，本書の第一版では，第四部の資料編にシステム共通域についての記述がありましたが，諸般の事情で第二版においてはこれをGRCG・EGCについての記述に差し替えてあります．あらかじめご了承下さい．

最後に，本書の執筆の話を持ちこんでくださり，さらにさまざまな相談に応じてくださった島さん他技術評論社の皆さんに，改めて深く謝意を表したいと思います．

1994年4月　東京理科大学EIC代表　　加藤　潔

PC-9801 PROGRAMMER'S BIBLE

CONTENTS

第一部　ハードウェアの概要　9

第二部　98各機能の標準的利用方法

第三部　H98・MATE編　427

第四部　資料編　445

カバーデザイン　八木ヨシユキ デザイン室　/　本文レイアウト　株式会社アビック

第１部
ハードウェアの概要

PROGRAMMERS BIBLE

§1-1 ブロックダイヤグラム

PC-9801シリーズ（以下98）のブロックダイヤグラム（システム構成図）は図1-1のようになっています．以下では，このブロックダイヤグラムの各部分について簡単に説明します．

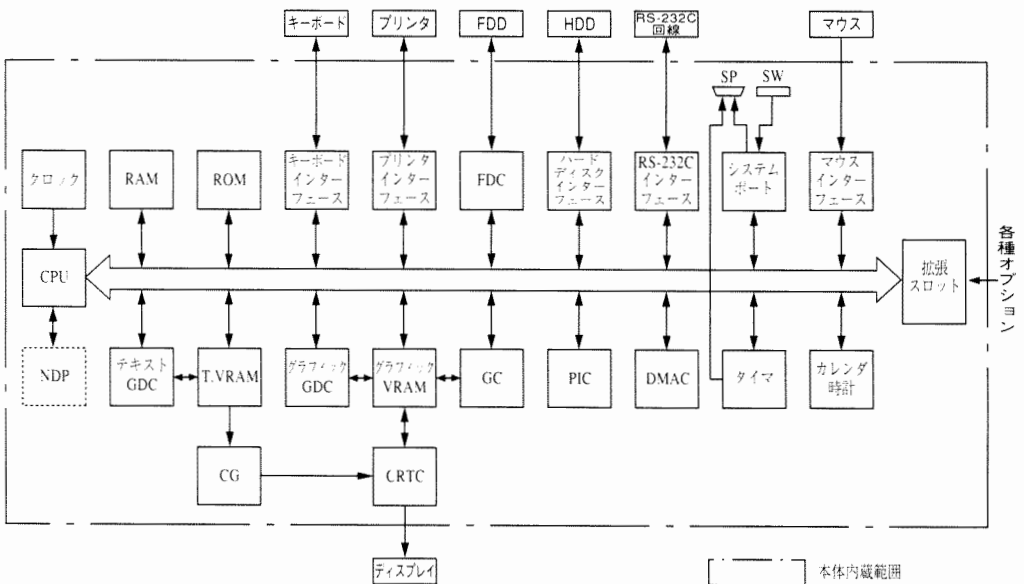



図1-1　ブロックダイヤグラム

●1．CPU

CPU(Central Processing Unit)は，命令の実行やシステム全体の制御を行う，システムの中核となるLSIです．98のCPUには，16ビット機には8086，V30，80286のうちのいずれか，32ビット機には80386，80486のうちのいずれかが採用されています．ただし，複数のCPUが搭載されている機種もあります．

●2．クロック

クロックは，CPUおよび周辺LSIを動作させるための基準クロックを供給します．周辺LSIに供給されるのがいわゆるシステムクロックですが，CPUに供給されるクロックを分周してシステムクロックを作っている機種と，CPUのクロックとシステムクロックが独立している機種があります．

●3．NDP（数値データプロセッサ）

NDP（Numeric Data Processor）は，CPUを補助するコプロセッサで，これが搭載されていると，CPU

の命令に浮動小数点数を扱う命令が追加されるので，数値計算が非常に高速になります．が，ほとんどの98でNDPはオプションとなっています．

●4．ROM

ROM(Read Only Memory)は，読み出しのみで書き込みができない代わりに，電源を切っても記憶内容が消えないメモリです．98のROMにはセルフチェックプログラムやIPL (Initial Program Loader), BIOS (Basic I/O System), N88-BASIC，機種によってはシステムセットアッププログラムやメニュープログラムなどが入っています．また，文字のパターンを記憶している文字パターンROMというものもあります．

●5．RAM

RAM(Random Access Memory)は，読み出し，書き込みが可能だが，電源を切ると記憶内容が失われるメモリで，単にメモリといえば普通RAMのことを指します．MS-DOSなどが動作しているリアルモードでは，普通に読み書きできるいわゆる“メインRAM”は最大640KB（キロバイト）です．メインRAM以外のRAMは“拡張RAM”と呼ばれ，主なメモリ管理方式にEMSやXMSという方式があり，これらの方式が定める手順に沿って読み書きを行います．

●6．キーボード

キーボードには，キーボード専用のLSIが内蔵されていて，キー入力の検出・キーリピートの付加・カナキーとCAPSキーの制御・本体との通信などを行っています．

●7．キーボードインターフェース

キーボードインターフェースは，キーボードと本体の間のデータのやり取りを制御しています．そのデータ転送形式は，RS-232C（●14参照）と同じ直列データ転送で，使われているLSIもRS-232Cと同等のものです．

●8．プリンタ

プリンタは，文字や図形を紙に印刷します．熱で文字を紙に転写する熱転写プリンタ，小さなハンマーでインクリボンをたたいて印刷するドットインパクトプリンタ，コピー機と同じ原理のページプリンタなどがあります．

●9．プリンタインターフェース

プリンタインターフェースは，プリンタへのデータの出力や，プリンタの状態の検出などを行います．データ転送形式は，セントロニクス方式という並列データ転送の一種です．

●10．FDD（フロッピーディスクドライブ）

FDD(Floppy Disk Drive)は，磁性体を塗った円盤（磁気ディスク）にデータを記録する記憶装置です．現在一般に使われている磁気ディスクは，大きさは5インチか3.5インチ，記録方式は容量1.2MB（メガバイト）の2HD，もしくは容量720KBの2DDです．

●11. FDC（フロッピーディスクコントローラ）

FDC(Floppy Disk Controller)は，フロッピーディスクの制御やディスクドライブとのデータのやりとりなどを行います．フロッピーからのデータをメモリに読み込んだり，逆にメモリのデータをフロッピーに書き込んだりするときには，DMAC（●29参照）と連携動作をすることでデータの高速転送を可能にしています．

●12. HDD（ハードディスクドライブ）

HDD(Hard Disk Drive)は，磁性体を塗った金属性の堅い円盤を，密封して高速回転させているディスクドライブです．フロッピーディスクよりもはるかに大容量で，かつ高速ですが，通常ディスクの取りかえはできません．

●13. ハードディスクインターフェース

ハードディスクインターフェースは，ハードディスクの制御やハードディスクドライブとのデータのやりとりなどを行います．初期のころは，NEC独自のインターフェース方式でしたが，最近ではSCSI（スカジー），SASI（サッシ），IDEなどの統一方式が採用されています．

●14. RS-232Cインターフェース

RS-232Cインターフェースは，汎用のインターフェースで，データを1ビットずつ直列にやり取りします．いろいろな周辺機器やモデム，他のコンピュータなどをつなぐことができます．

●15. システムポート

システムポートは，本体内部のさまざまなシステム情報をセットしたり読み出したりするときに使います．

●16. SP

スピーカです．

●17. SW

システム立ち上げ時にシステムプログラムが設定するシステムの動作モードを指定するスイッチで，ディップスイッチと呼ばれています．

●18. マウス

マウスは，メニュー選択や絵を描くときなどに使う入力装置です．

●19. マウスインターフェース

マウスインターフェースは，マウスからの入力データを加工してCPUに渡します．

●20. テキストGDC

テキストGDCは，文字を専門に表示するテキスト画面を制御します．

●21．テキストVRAM

テキストVRAM(Text Video RAM)は，テキスト画面に表示する文字の文字コードや色などを記憶するRAM領域です．CPUからだけでなく，表示のためにCRTコントローラからもアクセスされます．

●22．CG（キャラクタジェネレータ）

CG(Character Generator)は，文字パターンのデータを専用のROMとRAMに持っていて，表示系やCPUに文字パターンデータを供給したり，CPUからの文字パターンの書き込み（ユーザー定義文字の書き込み）を受けつけたりします．

●23．グラフィックGDC

グラフィックGDCは，グラフィック画面の制御や，CPUとは独立に直線・四角形・円の描画などを行います．なお，GDCとはGraphic Display Controllerの略なので，本来ならグラフィックGDCというのはおかしいのですが，GDCはテキスト画面の制御にも用いられているので，この呼び名があります．

●24．グラフィックVRAM

グラフィックVRAM(Graphic Video RAM)は，画面に表示するグラフィックデータを記憶するRAM領域です．98では，グラフィックVRAMのビットが1のとき点を表示し，0のとき点を表示しない“ビットマップ方式”というグラフィック表示方式がとられています．

●25．GC（グラフィックチャージャ）

GC(Graphic Charger)は，CPUからのデータを，加工しながら高速にグラフィックVRAMに書き込みます．98に搭載されているGCには現在のところ，GRCG，EGC，E^2GCの3種類があり，GCを搭載している機種ではこのうちのどれか1つが搭載されています．

●26．CRTディスプレイ

98のCRTディスプレイは，テレビなどと同じ原理のラスタ・スキャン方式という表示方式を採用しています．

●27．CRTC（CRTコントローラ）

CRTC(CRT Controller)は，テキスト画面とグラフィック画面の重ね合わせや，表示文字の形式の決定，CRTディスプレイの制御などを行います．

●28．PIC

PIC(Programmable Interrupt Controller)は，外部からの割り込み要求を取りまとめ，一括してCPUに割り込み要求を出します．

●29．DMAC（DMAコントローラ）

DMAC(Direct Memory Access Controller)は，CPUを介することなく直接データの転送を行います．主

に，外部記憶装置とメモリとの間のデータ転送に使われています．

●30．タイマ

タイマは，システムクロックをカウントするカウンタ／タイマで，CPUに定期的に割り込みをかけるときや，ブザー周波数の決定などに用いられます．RS-232Cインターフェースへのクロックの供給も行っています．

●31．カレンダー時計

カレンダー時計は，年，月，日，時，分，秒を持っていて，電源を消しても時をきざみ続ける時計です．

●32．拡張スロット

拡張スロットは，各種の拡張ボードを差すためのスロットです．

§ 1-2 CPU

CPU (Central Processing Unit) は，システム全体を制御し，実際の命令の実行を行うLSIです．CPUはシステムの中枢であり，CPUの性能がシステム全体の性能を大きく左右します．それだけに，コンピュータで効率的かつ互換性の高いプログラムを組むためには，CPUの特性をよく理解することが重要になります．そこでここでは，98シリーズに採用されているインテル製CPUの80x86，およびNEC製CPUのV30についての解説をしますが，これらについて詳しく解説すると，それだけで本が何冊も書けるほどの量になってしまいます．そこで，ここでは各CPUの主な特徴およびプログラムを組むときの注意点について述べるだけにとどめ，各CPUについての詳細な説明は関係書籍に譲ることにします．

■1-2-1 8086,V30

8086は，インテル製の16ビットCPUの最も初期のものです．その主な特徴は次の通りです．

●16ビットのデータバス幅

普通，「○○ビットCPU」というと○○はデータバス幅を表します．このデータバス幅が大きいほど，一度に処理できるデータの量が多くなるため，処理速度は高速になります．8086は，このデータバス幅が16ビットです．

●1Mバイトのメモリアドレス空間

8086は，メモリのアドレスバス幅が20ビットなので，1Mバイトまでのメモリを直接アクセスできます．しかし，最近のアプリケーションソフトは巨大なメモリ空間を必要とするものが多いので，この1Mバイトというメモリ空間はすでに窮屈なものになっています．

●プリフェッチキューの搭載

CPUの命令は，通常は実行する順番にメモリに格納されています．そこで8086は，ある命令を読み取ったあと，バスが空いていれば次の命令をあらかじめ読み込んでおき，プリフェッチキューというバッファの一種にためておきます．こうすることで，データの読み書きと命令の読み込みが衝突することが少なくなり，処理速度が向上しています．

●デバッグ機能のサポート

シングルステップ割り込みやブレークポイントなどのデバッグ機能をサポートしています．

●16/8ビット幅の汎用レジスタ

8086の汎用レジスタは16ビット幅ですが，8ビットCPUとの互換性のために，1本の16ビットレジスタ

を2本の8ビットレジスタとしても扱えるようになっています.

●バイト単位のアドレス指定

8086のアドレス指定はバイト単位（8ビット単位）であり，たとえワードデータ（16ビットデータ）でもバイト単位の任意のアドレスに格納することができます．ただし，16ビットデータを奇数アドレスに格納すると，アクセスが2回生じ，データの転送効率は低下します.

●リトルエンディアンのデータ格納形式

8086のデータ格納形式は，下位のデータほど前の方に格納されるリトルエンディアンの形式になっています.

V30は，NECが開発した8086上位互換のCPUであり，8086に比べて以下のような特徴を持っています.

●命令の所要クロック数の低減

8086ではマイクロプログラムで実現されていた命令の一部を，ワイヤードロジック化することなどによって，命令の実行に必要なクロック数を低減し，全体的な処理速度の向上を実現しています.

●拡張命令の追加

8086にはなかった命令が数個追加されています．が，これらの命令は80286以降のCPUではサポートされていないので，互換性を考慮するならこのV30独自の拡張命令は使わない方が無難です.

以上の点を除けば，V30は8086とほぼ同じであるといっていいでしょう．実際，8086を搭載している98の初期マシンのCPUをV30に差し替えても，ほとんど正常に動作するようです.

■1-2-2 80286

8086の次に開発されたのは80186，および80286でした．80186は，8086に周辺LSIを組み込むなどしたものでしたが，制御用などに使われた以外，あまり普及しませんでした．それに対し，80286は，メモリ管理などの高度な処理が可能なプロテクトモードを備え，8086に比べ格段に進歩したCPUで，パソコンなどに広く使用されています．その80286の主な特徴は次の通りです.

●プロテクトモードの搭載

80286は，8086と命令レベルで互換性があるリアルモードのほかに，メモリ管理やマルチタスクなどの高度な処理が行え，16Mバイトのメモリ空間にアクセスできるプロテクトモードを備えています.

●16Mバイトの物理メモリアドレス空間

80286はメモリのアドレスバス幅が24ビットに拡張されたので，8086の16倍の16Mバイトまでのメモリ空間を直接アクセスできます．ただし，1Mバイト以上のメモリ空間へのアクセスは，通常，80286本来の実力を発揮できるプロテクトモードでのみ可能です.

●パイプライン処理の採用

CPUの命令の処理には，命令の解釈・実行・結果の格納などのいくつかの段階があります．その各段階を完全に独立させ（その段階をステージという），各ステージを処理するユニットを同時平行に動作させることによって，命令の処理を高速化するのがパイプライン処理です．たとえば，ある命令の結果を格納しながら，次の命令の実行をする，などということをするわけです．このパイプライン処理により，80286は8086に比べ処理速度が大幅に向上しています．

■1-2-3 ——— 80386

80386は，80286のデータバス幅を32ビットにするとともに，レジスタのビット幅の拡張やメモリ空間の拡大，プロテクトモードの強化などを行ったものです．その主な特徴は次のようなものです．

●32ビットのデータバス幅

データバス幅が32ビットになったことで，一度に転送可能なデータの量は2倍になりました．ただし，既存の16ビットデータバスに対応したソフトを使っているかぎり，データの転送速度は2倍にはならず，せいぜい25％程度の転送速度の向上しか望めません．

●4Gバイトの物理メモリアドレス空間

80386ではアドレスバス幅が32ビットまで拡張され，4Gバイトまでのメモリに直接アクセスすることが可能です．ただし，80286と同様に，1Mバイト以上のメモリ空間にアクセスできるのは通常プロテクトモード時のみです．

●32/16/8ビット幅の汎用レジスタ

データバス幅が32ビットになったのに伴い，汎用レジスタのビット幅も32ビットに拡張されています．ただし，これまでのCPUとの互換性を保つため，これらを16/8ビットのレジスタとしても扱えるようになっています．

●プロテクトモードの強化

80386のプロテクトモードは，80286のプロテクトモードを含みつつ拡張され，いくつかの命令が追加されています．

●仮想86モードの搭載

80386では，8086のアプリケーションをプロテクトモードの1タスクとして実行でき，仮想マシンを実現できる仮想86モードが新たに加えられました．

■1-2-4 ——— 80486

80486は，一部にRISC(Reduced Instruction Set Computer，縮小命令セットコンピュータ）の技術を適用することによって，実行速度を飛躍的に向上させたCPUです．その主な特徴は次のようなものです．

●命令の所要クロック数の低減

80486は，RISC技術の適用により，よく使われる命令は1クロックで実行を終了します．そのほかの少し複雑な命令も所要クロック数は低減されているので，全体としての処理速度は非常に高速になっています．ただし，若干かえって所要クロック数が増加した命令もあります．

●キャッシュメモリの搭載

CPUの処理速度を向上させるうえで大きな障害となっている要因の1つに，外部メモリの低速さがあります．そこで，低速な外部メモリへのアクセスをできるだけ減らすため，キャッシュメモリという一種のバッファを設けておいて，過去に読み込まれたデータをそのキャッシュメモリにためておき，CPUがアクセスすべきデータがキャッシュに存在していれば，外部メモリではなくそちらをアクセスするという手法がよく使われます．80486は，このキャッシュメモリを8Kバイト，CPUに内蔵しています．

●命令の追加

キャッシュコントロール命令など，いくつかの命令が追加されています．

なお，80x86は基本的に，その前までに出ていた80x86の機能を完全に含んでいます（たとえば80386は8086，80286の機能を完全に含んでいる）．

■1-2-5 各CPU間の相違点

80x86は前述のように，基本的にはその前までの80x86を完全に含む形で拡張されてきましたが，特に8086から80286に移る際に，細かい点で互換性に影響が出てくるような変更が加えられています．以下に，そのような互換性が問題になるような変更がされた点と，それぞれの変更点についての互換性を保つための処置を列挙し，さらに，互換性が問題にはなりにくいが，CPUの判別をするのに便利な相違点についても述べます．

●命令クロック数，および命令処理方式の相違

8086(V30)と80286以降のCPUでは，ほとんどの命令について80286以降のCPUの方が少ないクロック数で実行が完了します．また，80286以降のCPUはパイプライン処理を行っているので，同じ命令を同じCPUで実行しても処理時間が著しく異なることがあり得ます．そのため，I/O制御などのタイミングが重要な処理に関しては，これらのことを考慮に入れてプログラムを組む必要があります．パイプライン処理による命令の実行速度の変化を防ぐには，JMP $+2命令など でパイプラインをクリアしてやればよいでしょう．

●除算命令のエラー割り込みの相違

DIV命令を実行した結果，オーバーフローなどで除算エラー割り込みが発生した場合，割り込み処理ルーチンでIRET命令を実行すると，80286以降のCPUでは例外が発生したDIV命令に戻るのに対し，8086（V30）では例外が発生したDIV命令の次の命令に戻ります．したがって，すべての80x86で動作で

きるような除算エラー割り込みルーチンを作るには，CPUが8086（V30）か80286以降のCPUかを判定し，スタックに積まれている戻り番地を操作してやる必要があります．

●例外処理の相違

80286以降のCPUは，8086（V30）よりも不正な命令やアクセスに対する例外の検出が厳しくなっています．たとえば，オフセットアドレスFFFFH（セグメントの最後尾）に対してワード（16ビット）アクセスを行うと，8086（V30）では何らかのアクセスを行って次の命令に進みますが，80286以降のCPUでは無効命令例外(INT6)が発生し，98の場合は通常ハングアップしてしまいます．したがって，互換性を保つためには，そのような不正なアクセスをしないように十分注意してプログラムを組む必要があります．

●未定義命令の相違

POP CSなどの未定義命令は，8086（V30）では一応動作するものもありましたが，80286以降のCPUでは別の命令が割り当てられていたり，例外割り込みが発生したりします．いずれにせよ，未定義命令は使うべきものではありません．

●PUSH SP命令の相違

PUSH SP命令を実行すると，80286以降のCPUではPUSH命令によって更新される前のSPの値がスタックに積まれるのに対し，8086（V30）では更新されたあとのSPの値がスタックに積まれるので，8086（V30）の場合，80286以降のCPUに比べて2だけ小さい値がスタックに積まれることになります．従って，互換性を考慮するなら，スタックに積まれたSPの値自体を参照するような使い方は，できるだけ避けるべきです．

なお，この相違点は，80286以降のCPUと8086（V30）の判別をしたいときに使うと便利です．

●多ビットシフト・ローテートの相違

80286以降のCPUでは，シフト命令で大きなシフト数が指定された場合，下位5ビット（最大31回のシフト）のみが有効でそれより上位のビットは無視されますが，8086（V30）は指定された通りの回数のシフトを行います．ローテート命令の場合も同様です．たとえば，シフト数として32を指定すると，80286以降のCPUでは上位のビットが無視されて1回もシフトが行われないのに対し，8086（V30）では32回のシフトが実行されます．

●IDIV命令の相違

80286以降のCPUでは，演算結果として最大の負数（80H，8000H）を返すことができますが，8086（V30）では除算エラー割り込みが発生します．従って，8086（V30）の場合，必要なら除算エラー割り込みルーチンで適当な処置を取るようにします．

●NDPの割り込み方式の相違

NDP（数値データプロセッサ）の割り込み要求は，80286以降のCPUではPIC（割り込みコントローラ）を介さず直接CPUに出されますが，8086（V30）ではPICを通して割り込み要求がなされます．したがって，8086（V30）でNDPの割り込みを使う場合には，PICのコントロールも必要になります．また当然のことながら，NDPの割り込みベクタ番号も異なったものになります．

●命令長の制限

8086（V30）には1命令のバイト数に制限はありませんが，80286は1命令の最大長は10バイト，80386は15バイトで，それ以上の命令長の命令があると無効命令例外（INT6）が発生します．が，通常は1命令が10バイトを超えることはまずありませんから，この制限は特に意識する必要はないでしょう．

●LOCKプリフィクスの適用範囲の相違

LOCKプリフィクスは，8086（V30）ではすべての命令に付けることが可能でしたが，80286以降のCPUでは一部の命令でしか有効ではなくなりました．が，98のシステム構成ではLOCKプリフィクスを用いることはまずないでしょうから，これも特に問題にはならないでしょう．

●フラグレジスタの動作の相違

80286と80386の間には，互換性が問題になるような違いはほとんどありませんが（ただし，80286のプログラムを80386で動作させる場合．逆はできない場合が多々ある），フラグレジスタの動作に違いがあります．80386以降のCPUには，フラグレジスタのビット14にNTというフラグが存在し，その部分を書きかえることができますが，80286以前のCPUではここは0または1に固定されています．NTフラグはプロテクトモードでのみ意味を持つフラグなので，リアルモードや仮想86モードでCPU判別のためにこのフラグを一時的に変更しても問題は生じません．したがって，32ビットレジスタ・仮想86モード・80386の拡張命令などの80386以降のCPUの機能が使えるかどうかを判別するには，このNTフラグが書きかえられるかどうかを調べてやればよいのです．

■サンプルプログラム

このサンプルプログラムは，以上に述べてきた相違点を利用して，CPUの種類が，8086（V30），80286，80386以降の3種類のうちのどれであるかを判別するプログラムです．まず，PUSH SPの相違を使って8086（V30）を判別し，次にフラグレジスタの相違を使って80286と80386以降の判別をしています．

```
CODE        SEGMENT  WORD                     ; ─┐ セグメントの定義
            ASSUME   CS:CODE,DS:CODE          ; ─┘
        ;
            ORG      100H                     ; ── ０１００Ｈから開始
        ;
START:      JMP      CHKCPU                   ; ── プログラムの先頭へジャンプ
        ;
MES8086     DB       "ＣＰＵは８０８６（Ｖ３０）です。$"   ; ─┐
MES80286    DB       "ＣＰＵは８０２８６です。$"           ;  ├─ メッセージ
MES80386A   DB       "ＣＰＵは８０３８６以降です。$"       ; ─┘
        ;
CHKCPU:     MOV      AX,CS                    ; ─┐ データセグメントのセット
            MOV      DS,AX                    ; ─┘
            PUSH     SP                       ; ─┐
            POP      AX                       ;  ├─ ８０８６と８０２８６以降の判別
            CMP      AX,SP                    ;  │  （本文参照）
            JZ       CP80286A                 ; ─┘
            MOV      DX,OFFSET  MES8086       ; ─┐ ＣＰＵが８０８６であるというメ
            JMP      DISPMES                  ; ─┘ ッセージを表示するためジャンプ
```

```
CP80286A:                                  ; 以下，286と386を判別
            PUSHF                          ; ―― フラグレジスタの保存
            PUSHF                          ; ┐
            POP     AX                     ; ├ AX，DX←フラグレジスタ
            MOV     DX,AX                  ; ┘
            XOR     AX,4000H               ; ―― NTフラグの反転
            PUSH    AX                     ; ┐ フラグレジスタ←AX
            POPF                           ; ┘
            PUSHF                          ; ┐ AX←フラグレジスタ
            POP     AX                     ; ┘
            POPF                           ; ―― 保存しておいたフラグの復帰
            CMP     AH,DH                  ; ┐ NTフラグが変化しているか判定
            JNZ     CP80386A               ; ┘ （変化なしなら80286）
            MOV     DX,OFFSET  MES80286    ; ┐ CPUは80286であるという
            JMP     DISPMES                ; ┘ メッセージのセット
CP80386A:   MOV     DX,OFFSET  MES80386A   ; ―― CPUは80386以降
        ;
DISPMES:    MOV     AH,09H                 ; ┐ 文字列表示のファンクションコー
            INT     21H                    ; ┘ ル
        ;
            MOV     AH,4CH                 ; ┐ DOSへのリターン
            INT     21H                    ; ┘
        ;
CODE        ENDS                           ; ┐ セグメントの終了（STARTか
            END     START                  ; ┘ ら実行開始指定）
```

§1-3 I/Oポート

I/Oポートは，CPUが周辺のLSIなどを制御したり，周辺のLSIからの情報を受け取ったりするときの窓口となるものです．タイマ，GDC，FDCなどの周辺LSIに対する入出力は，ほとんどこのI/Oポートを介して行われます．

I/Oポートには，メモリと同じようにアドレスがあります．以下，このアドレスのことをI/Oアドレスと呼ぶことにします．周辺LSIは，I/Oアドレスの特定の番地に割り当てられていて（通常複数のアドレスを占める），CPUはそのI/OアドレスのI/Oポートに対して読み書きを行うことによって，そのLSIを制御します．

■1-3-1 I/Oポートの使用例

80x86のCPUには，　I/Oポートを介してデータの入出力を行うために，IN命令，OUT命令という命令が用意されています．IN命令は，CPUがI/Oポートからデータを入力する命令，OUT命令は，CPUがI/Oポートにデータを出力する命令です．それに対応して，C言語にもI/Oポートに対してのデータの入出力をする命令があります．データを入力するinportb命令，データを出力するoutportb命令です．

例を挙げましょう．I/Oポートの31H，33H，37Hにはシステムポートが割り当てられていますが，そのうちの37Hに06Hを書き込むとブザーが鳴り出し，同じところに07Hを書き込むとブザーが止まります．そこで，アセンブラでブザーを鳴らしたいときは，

```
MOV AL, 06H
OUT 37H, AL
```

逆にブザーを止めたいときには，

```
MOV AL, 07H
OUT 37H, AL
```

とすればよいのです．また，同様に，C言語でブザーを鳴らしたいときには，

```
outportb(0x37,0x06);
```

ブザーを止めたいときは，

```
outportb(0x37,0x07);
```

とすればよいことになります．

もう1つ例を挙げましょう．今，マウスの左ボタンが押されているか調べたいとします．そのとき，参照すべきI/Oポートは7FD9Hのマウスインターフェース・ポートAです．ところが，IN命令，OUT命令ともに，命令の中で直接指定できるアドレスは00H～FFH（8ビット幅）なので，今の場合，上の例のように直接アドレスを指定することはできません．そこで，アセンブラでマウスボタンの状況を調べるには，アドレスをいったんDXレジスタに入れて，

```
MOV DX, 7FD9H
IN AL, DX
```

とします．すると，ALレジスタにマウスからのいろいろな情報が入力されます．そのうち，左ボタンの状況はビット7にセットされているので，

```
TEST AL, 80H
```

とします．こうすれば，左ボタンが押されていればゼロフラグが1になります．

同様に，C言語でマウスの左ボタンの状態を調べるとすると，C言語ではアドレス指定は最初から0000H～FFFFH（16ビット幅）なので，特別な配慮は必要なく，

```
int  a;
a=inportb(0x7fd9);
```

とすれば変数aにマウスの情報が入ります．そのビット7を調べるために，

```
a = a & 0x80;
```

すれば，左ボタンが押されているときにはaに0が，離されているときにはaに0x80が入ります．

■1-3-2――I/Oアクセス時のウェイト

I/Oポートに接続されている周辺LSIの中には，8ビットCPU時代のLSIを中心に，処理速度が遅いためにCPUによるウェイトなしの連続アクセスに耐えられないものがあります．そのようなLSIに対して連続してデータを入出力する場合には，ソフトウェア的にウェイトを取ってやる必要があります．そこで，どうやってウェイトを取るかですが，パイプライン処理による実行速度の変化の影響を受けにくく，かつ何の処理も行わないJMP $+2をいくつか挿入することでウェイトを取ってやるのが最適です．具体的に，どのLSIにどれだけのウェイトが必要かについては，4-4の「I/Oアクセス時の必要ウェイト数」を参照してください．

§ 1-4 割り込み

80x86の割り込みには，ハードウェア割り込みとソフトウェア割り込みの2種類がありますが，そのいずれもがソフトウェアを作成する上で非常に重要な概念です．そこでここでは，この2種類の割り込みについてできるだけ詳しく説明することを試みます．

■1-4-1 ハードウェア割り込み（外部割り込み）

ハードウェア割り込みというのは，キーボードのキーが押された，RS-232Cにデータが届いた，などといったCPU外の要因をきっかけとして，CPUが特定のルーチンを実行することをいいます．

表1-1（1） PIC関係のI/Oポート【マスタ】

リード／ライト	I／O アドレス	機能	データ							
			D7	D6	D5	D4	D3	D2	D1	D0
リード	00H	ポールモードの読み出し	I	×	×	×	×	W2	W1	W0
	00H	IRRの読み出し	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
	00H	ISRの読み出し	IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0
	02H	IMRの読み出し	M7	M6	M5	M4	M3	M2	M1	M0
ライト	00H	ICW1の書き込み	0	0	0	1	LTIM	0	SNGL	1
	02H	ICW2の書き込み	T7	T6	T5	T4	T3	0	0	0
	02H	ICW3の書き込み	1	0	0	0	0	0	0	0
	02H	ICW4の書き込み	0	0	0	SFNM	1	1	0	1
	02H	OCW1（IMR）の書き込み	M7	M6	M5	M4	M3	M2	M1	M0
	00H	OCW2の書き込み	R	SL	EOI	0	0	L2	L1	L0
	00H	OCW3の書き込み	0	ESMM	SMM	0	1	P	RR	RIS

外部からの割り込み要求を受けると，CPUは，フラグレジスタの中のインタラプトフラグを参照して，現在外部割り込みが許可されているかどうかを調べます．許可されていれば，今まで行っていた処理を中断し，絶対番地00000Hから1Kバイトあるベクタテーブルという領域の，どこからの割り込みかによって決まっている場所を参照します．ベクタテーブルには，割り込み処理ルーチンのアドレスがオフセット，セグメントの順に書かれていて，CPUはそのアドレスに対して特殊なサブルーチンコールを行って，割り込み処理を行います．そして，割り込み処理が終わると，何事もなかったかのようにそれまで行っていた処理を再開します．

このような外部からの割り込み要求を取りまとめ，一括してCPUに対して割り込み要求を出すのがPIC（割り込みコントローラ）です．98のPICは，8259AというLSIの相当品です．98は，このPICを2個使って，最大14までの周辺LSIからの割り込み要求を受けられるようになっています．

2個の8259Aを制御するI/Oポートを，表1-1に示します．

98の2個のPICは，図1-2のような構成になっています．2個のPICは，それぞれマスタPIC・スレーブPICと呼ばれていて，マスタPICはCPUに直接割り込み要求を出すのに対し，スレーブPICは，いったんマスタPICに割り込み要求を出し，マスタPICにCPUへの割り込みをかけてもらいます．このようなPICの接続のしかたを，カスケード接続といいます．

この，PICを通した各LSIからの割り込みを利用したプログラムを作るときには，次の3種類のルーチンを用意する必要があります．

表1-1（2）　PIC関係のI/Oポート【スレーブ】

リード／ライト	I／O アドレス	機能	D7	D6	D5	D4	D3	D2	D1	D0
リード	08H	ポールモードの読み出し	I	×	×	×	×	W2	W1	W0
	08H	IRRの読み出し	IR15	IR14	IR13	IR12	IR11	IR10	IR9	IR8
	08H	ISRの読み出し	IS15	IS14	IS13	IS12	IS11	IS10	IS9	IS8
	0AH	IMRの読み出し	M15	M14	M13	M12	M11	M10	M9	M8
ライト	08H	ICW1の書き込み	0	0	0	1	LTIM	0	SNGL	1
	0AH	ICW2の書き込み	T7	T6	T5	T4	T3	0	0	0
	0AH	ICW3の書き込み	0	0	0	0	0	1	1	1
	0AH	ICW4の書き込み	0	0	0	SFNM	1	0	0	1
	0AH	OCW1（IMR）の書き込み	M15	M14	M13	M12	M11	M10	M9	M8
	08H	OCW2の書き込み	R	SL	EOI	0	0	L2	L1	L0
	08H	OCW3の書き込み	0	ESMM	SMM	0	1	P	RR	RIS

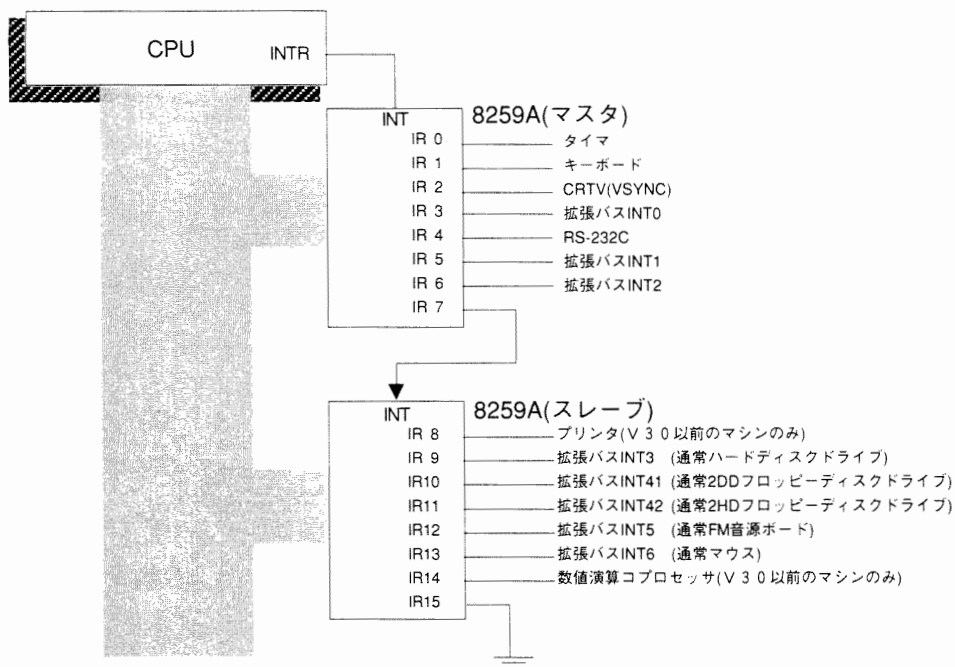



図1-2　PICの構成

①割り込み処理ルーチン
②割り込み組み込みルーチン
③割り込み切り離しルーチン

①の割り込み処理ルーチンというのは，割り込みがかかったときに行う処理を記述したルーチン，②の割り込み組み込みルーチンというのは，目的の割り込みがかかるようにし，割り込みがかかったときに割り込み処理ルーチンに処理が移るようにするルーチン，さらに③の割り込み切り離しルーチンというのは，ある割り込みをかからなくし，その割り込みの使用を終了させるルーチンのことです．

割り込み処理の手順としては，まず割り込み組み込みルーチンで割り込みを開始させ，割り込みがかかるたびに割り込み処理ルーチンで必要な処理を行い，プログラムの終了時など割り込み処理が必要なくなったときには，割り込み切り離しルーチンによって割り込みの使用を終了させるというのがごく普通の手順です．以下に，これら3種類のルーチンのそれぞれについて，行うべき処理を順に述べていきます．

①割り込み処理ルーチン

1）外部割り込みを許可する

外部割り込みがかかると，CPUは割り込み処理ルーチンを呼び出す前に，自動的に外部割り込みを禁止し，何もしないと割り込み処理ルーチンを実行している間ずっと割り込み禁止状態のままになりま

す．そうなると不都合なことが起こる場合があるので，通常は，自分より優先順位が高い割り込みを許可するために，割り込み処理ルーチンの先頭にはまずSTI命令を置きます．

2）使用するすべてのレジスタを保存する

割り込み処理ルーチンは，処理が終わったあと，割り込みがかかる前にどんな処理を行っていたとしてもその処理を正常に続行できるように作らなければなりません．そのため，割り込み処理ルーチンで使用するレジスタはすべて保存しておく必要があります．ただし，フラグレジスタはCPUによって自動的に保存されるため，保存する必要はありません．

3）割り込み処理を行う

目的とする割り込み処理を行います．

4）PICにEOIを送出する

割り込み処理が終了したら，そのことをPICに通知するためにPICにEOI（End Of Interrupt）を送ってやります．具体的には，マスタPICの割り込みの場合には，

```
MOV   AL,20H
OUT   00H,AL
```

としてやります．スレーブPICの場合には少し複雑で，スレーブPICにEOIを送るのはもちろんですが，スレーブPICの割り込みがすべて終了していたとき（スレーブPICのISR（Interrupt Service Register）が0だったとき）にはマスタPICにもEOIを送ってやる必要があります．具体的には以下のようにします．

このようにしてEOIを送ると，優先順位が自分以下の割り込みも許可されます．したがって，割り込み処理中にも優先順位が自分以下の割り込みも許可したい場合には，処理を始める前にEOIを送り出すようにします．ただし，その場合，その割り込み処理中に同じ割り込みがかかってしまう，いわゆる“再入”が起こらないように十分注意する必要があります．

5）保存しておいたレジスタを復帰する

2）で保存しておいたレジスタを元に戻します．

6）IRET命令で割り込みルーチンを終了する

割り込み処理ルーチンを終了して元の処理に戻るときには，通常のRET命令ではなくIRET（インタラプトリターン）命令を実行します．

第一部 ハードウェアの概要 §1・4 割り込み

②割り込み組み込みルーチン

1）CLI命令によって外部割り込みを禁止する

割り込みがかかるようにするには，PICやベクタテーブルの設定を行う必要がありますが，これらを途中まで設定したところで割り込みがかかってしまうことを防ぐため，最初に外部割り込みを禁止します．

2）ベクタテーブルに割り込み処理ルーチンのアドレスをセットする

ベクタテーブルの目的の割り込みに相当する部分に，割り込み処理ルーチンの先頭アドレスを書き込んでやります．そのとき，書きかえる前に書いてあった値は保存しておきます．

3）割り込み元になる周辺LSIの設定を行う

実際に割り込み要求を出す周辺LSIの設定を行います．ここでいう設定とは，たとえばタイマ割り込みならカウントモードとカウント数のセット，CRTV（VSYNC）割り込みならCRT割り込みリセットの発行，などといった処理のことです．

4）PICのIMRの所定のビットをクリアする

PICの内部には，割り込み要求を個別に許可・禁止するためにIMR（インタラプトマスクレジスタ）というレジスタが存在しています．IMRの1ビットが1つの割り込みに対応していて，そのビットが0のとき割り込み許可，1のとき割り込み禁止となります．このIMRによる個別の割り込み禁止を解除するために，IMRの目的の割り込みに相当するビットをゼロクリアしてやります．

5）STI命令によって外部割り込みを許可する

STI命令によって外部割り込みを許可します．これによって，以後割り込みがかかるたびに割り込み処理ルーチンが呼び出されるようになります．

③割り込み切り離しルーチン

1）CLI命令によって外部割り込みを禁止する．

割り込み組み込みルーチンのときと同じ理由で，まず外部割り込みを禁止します．

2）PICのIMRの所定のビットをセットする．

PICのIMRの対象となっている割り込みに相当するビットをセットし，PICによる割り込みの個別の許可・禁止を禁止に設定します．あるいは，割り込み組み込みルーチンで書きかえる前のIMRを保存しておき，ここでそれを復帰させてもいいでしょう．

3）保存しておいたベクタテーブルの以前の値を復帰する．

ベクタテーブルの値を，組み込みルーチンによって書きかえられる前の状態に戻します．

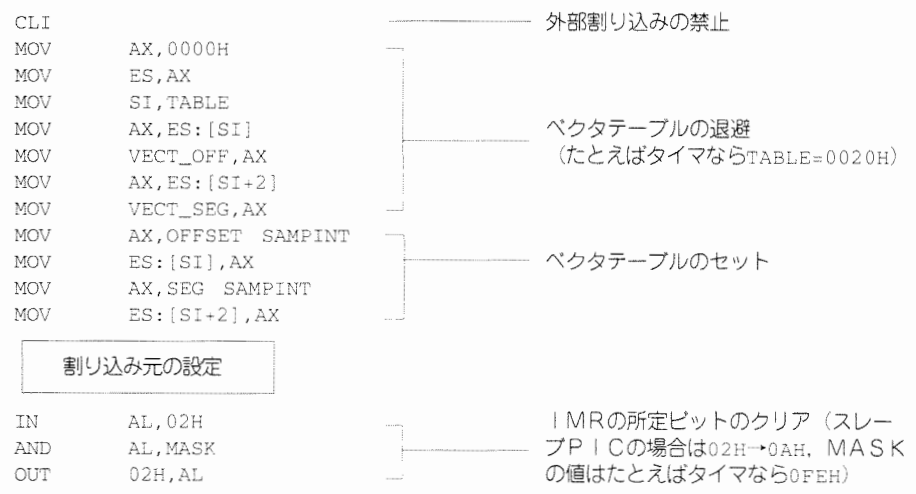
4）STI命令によって外部割り込みを許可する．

以後の他の割り込みを許可するために，外部割り込みを許可します．

■プログラム構成例

以上に述べてきたことを踏まえて，割り込み処理ルーチンと，割り込み組み込みルーチン，および割り込み切り離しルーチンの構成例を示します．

1）割り込み処理ルーチンの構成例（マスタPICの場合）

2）割り込み組み込みルーチンの構成例（マスタPICの場合）

```
JMP     $+2          ―― パイプラインのクリア
STI                  ―― 外部割り込みの許可
```

なお，各ラベルの意味は次の通りです．

- SAMPINT　：割り込み処理ルーチンの先頭を示すラベル
- MASK　：PICのマスクレジスタの目的の割り込みに相当する位置のビットだけが0になっている数値
- TABLE　：目的の割り込みのベクタテーブルのオフセットアドレス
- VECT_OFF, VECT_SEG　：ベクタテーブルを保存しておく変数（DWで定義）

これらのうち，MASKとTABLEの値は，割り込みの種類によって違いますが，これらの値を割り込みごとにいちいち考えたり調べたりするのは面倒なことなので，各割り込みについてのMASKとTABLEの値を表1-2にまとめて示しておきます．

3）割り込み切り離しルーチンの構成例（マスタPICの場合）

なお，ラベルの意味は2）と同じです．また，IMRの所定ビットのセットの部分は，MASKの値が最初からわかっているなら（そういう場合が大部分ですが），

```
IN      AL,02H
OR      AL,(MASKを反転した数値)
OUT     02H,AL
```

としてもOKです（マスタPICのとき，スレーブPICのときは02H→0AH）．

表1-2　IMRをマスクする数値とベクタテーブルアドレス一覧

割り込み名	要求先のＰＩＣ	割り込みレベル	MASKの値	ベクタ番号	TABLEの値
タイマ	マスタ	ＩＲ０	0FEH	０８Ｈ	0020H
キーボード	マスタ	ＩＲ１	0FDH	０９Ｈ	0024H
ＣＲＴＶ（ＶＳＹＮＣ）	マスタ	ＩＲ２	0FBH	０ＡＨ	0028H
拡張バス　ＩＮＴ０	マスタ	ＩＲ３	0F7H	０ＢＨ	002CH
ＲＳ－２３２Ｃ	マスタ	ＩＲ４	0EFH	０ＣＨ	0030H
拡張バス　ＩＮＴ１	マスタ	ＩＲ５	0DFH	０ＤＨ	0034H
拡張バス　ＩＮＴ２	マスタ	ＩＲ６	0BFH	０ＥＨ	0038H
スレーブＰＩＣ	マスタ	ＩＲ７	07FH	------	------
プリンタ（Ｖ３０以前）	スレーブ	ＩＲ８	0FEH	１０Ｈ	0040H
拡張バス　ＩＮＴ３（ハードディスク）	スレーブ	ＩＲ９	0FDH	１１Ｈ	0044H
拡張バス　ＩＮＴ４１（2DD FD）	スレーブ	ＩＲ１０	0FBH	１２Ｈ	0048H
拡張バス　ＩＮＴ４２（2HD FD）	スレーブ	ＩＲ１１	0F7H	１３Ｈ	004CH
拡張バス　ＩＮＴ５（ＦＭ音源）	スレーブ	ＩＲ１２	0EFH	１４Ｈ	0050H
拡張バス　ＩＮＴ６（マウス）	スレーブ	ＩＲ１３	0DFH	１５Ｈ	0054H
ＮＤＰ（Ｖ３０以前）	スレーブ	ＩＲ１４	0BFH	１６Ｈ	0058H
ノイズ（無接続）	スレーブ	ＩＲ１５	07FH	１７Ｈ	005CH

■1-4-2　ソフトウェア割り込み（内部割り込み）

ソフトウェア割り込みというのは，CPUがINT命令という命令を実行することによって発生する割り込み，ということができます．CPUの命令によって特定のルーチンをコールするわけですから，ソフトウェア割り込みは，機能的には，割り込みというよりもむしろFAR CALL命令に近いものがあります．実際，多くの場合ソフトウェア割り込みはFAR CALL命令と同じ目的，つまりセグメントを超えたサブルーチンの呼び出しのために用いられます．

では，ソフトウェア割り込みはFAR CALL命令とどう違うかというと，まず，INT命令ではサブルーチンのアドレスを直接指定せず，ベクタ番号という0～255の番号を指定する，ということがあります．CPUは，このベクタ番号を元にしてハードウェア割り込みのところで出てきた　　　　ベクタテーブルを参照し，そこに書かれているアドレスに対してコールを行います．実際には，指定されたベクタ番号を4倍したアドレスの部分のベクタテーブルが参照されます（1つのサブルーチンの先頭アドレスがオフセット，セグメント合わせて4バイトであるため）．このことによって，ソフトウェア割り込みを使えば，たとえコールしたいルーチンの絶対アドレスを知らなくても，あるいはルーチンの絶対アドレスがシステムの状況によって変わっていても，ベクタテーブルが正しくセットされていて，ベクタ番号を正しく指定すれば，ターゲットとしているルーチンをコールすることができるのです．

このようなソフトウェア割り込みの利点のために，98では，システムの基本入出力プログラム（BIOS, Basic I/O System）やDOSのファンクションコールなどを呼び出すのに，このソフトウェア割り込みを利用しています．これらのシステムプログラムを呼び出すに当たっては，これらを呼び出すためのベクタテーブルはすでにシステムによってセットされているので，ユーザーは決められたベクタ番号を指定してINT命令を実行するだけで済みます．

INT命令のそのほかの特徴としては，INT命令は，コールするときに外部割り込みと同じようにフラグレジスタをプッシュし，外部割り込みを禁止するということがあります．また，元のルーチンにリターンする命令もIRET命令です．そのため，INT命令で呼び出されるルーチンでは通常，先頭で外部割り込みを許可し，結果をフラグレジスタに返したいときには，スタックに積まれているフラグレジスタの値を操作するようにします．

§1-5 DMA

DMAというのは，CPUを介することなく直接メモリに対する読み書きを行うことをいい，98では主にディスクドライブなどの外部記憶装置とのデータのやり取りにこのDMAが用いられています．98では，DMAのコントロールのために8237AというLSIの相当品を搭載していますが，このLSIの詳しい解説や98での実際のDMA転送のしかたについて述べると非常に煩雑になってしまいます．また，実際には一般ユーザーがDMAコントローラを直接制御してDMA転送を行うことは，まずないと思われますので，ここではDMA転送の具体的方法については触れず，DMAを用いるBIOS（具体的にはディスクBIOS等）を使うときに問題になるような8237Aの特徴とDMAに関する制限事項について述べることにします．

98でDMAを使うときにまず注意しなければならないのは，98のDMAコントローラ8237Aは，基本的に8ビットCPU（i8085等）のためのLSIである，ということです．i8085はメモリ空間が64Kバイトしかありませんでしたから，8237Aは64Kバイトのメモリ空間内でのDMA転送しか想定していません．では，98では，このLSIでどうやって80x86の1Mバイトまたはそれ以上のメモリ空間に対してのDMA転送を実現しているかというと，メモリ空間を1区画64KバイトのDMAバンクという領域に分割し，どのDMAバンクに対してDMA転送を行うかをあらかじめ指定しておいてからDMA転送を行う，という方式を取っているのです．

以上のことから，98のDMA転送には次のような制限がつきます．

1）一度に64Kバイト以上のDMA転送はできない．

2）複数のDMAバンクにまたがるDMA転送はできない．つまり，10000Hで割り切れる絶対アドレスを中間に含むようなDMA転送はできない（ただし80286以上搭載機種では可能）．

このうち，1）についてはそれほど大きな問題にはならないでしょうが，問題は2）で，この制限がつく機種で，BIOSによるリロケータブルなデータ領域に対するディスクロード等を行う場合には，この点を十分に注意する必要があります．

さらに，98のハード構成から次のような制限がつきます．

3）TVRAM，GVRAM，CGウィンドウなどの特殊なメモリに対するDMA転送は避けるべきである．

この，避けるべきである，というのは，そのような領域にDMAを行えば必ず不都合が起きる，という意味ではなく，そういうことをすると少し不安がある，ぐらいの意味です．実際，VRAMに対してDMA転送を行っても，多くの場合何の不都合も生じません．しかし，やはりこのような特殊なメモリに対するDMA転送は，できるだけ控えた方がいいでしょう．

§ 1-6 ディップスイッチ・メモリスイッチ

ディップスイッチとメモリスイッチは，システムの立ち上げ時にシステムプログラムによって設定される，システム各部の動作モードの選択のために用いられます．昔の機種では，ディップスイッチは機械スイッチ，メモリスイッチは不揮発性メモリ*に記録されている情報でしたが，今では一部を除いて機械スイッチによるディップスイッチはなくなり，ディップスイッチの設定内容もメモリスイッチとは別の不揮発性メモリ領域に保存されるようになっています．以下に，各ディップスイッチとメモリスイッチの意味と，その書き換え・参照方法を示します．

（※）バッテリでバックアップされていて，電源を切っても記憶が消えないメモリ．

■1-6-1 ディップスイッチ

ディップスイッチの設定内容の意味を表1-3，表1-4，表1-5に示します．なお，この表では一部の機種でしか意味を持たないスイッチは省略してあります．また，ここに挙げたスイッチの意味のうちで，機種によっては無効なものもあります．詳しくは各機種のマニュアルを参照してください．

表1-3　ディップスイッチSW1の意味

スイッチ名	番号	目的	ON	OFF	ソフトからの参照手段
SW1	1	ディスプレイ解像度の選択	専用高解像度（水平周波数24KHz）	標準解像度（水平周波数15KHz）	システムポートの参照
	2	スーパーインポーズ機能の選択	スーパーインポーズ機能を使用する	スーパーインポーズ機能を使用しない	
	3	プラズマディスプレイ使用の指定	プラズマディスプレイを使用する	プラズマディスプレイを使用しない	プリンタポートの参照
	4	ＦＤのドライブ番号の指定	内蔵ＦＤ #3,#4 外部ＦＤ #1,#2	内蔵ＦＤ #1,#2 外部ＦＤ #3,#4	
	5, 6	ＲＳ－２３２Ｃの同期モードの指定	スイッチ５　スイッチ６　（＊＊） ＯＮ　ＯＮ　ＢＣＩ同期モード ＯＮ　ＯＦＦ　ＳＴ２同期モード ＯＦＦ　ＯＮ　同期刻時機構 ＯＦＦ　ＯＦＦ　調歩同期モード		マウスポートの参照
	7				
	8	グラフィック処理ルーチンのモード指定	拡張グラフィックモード（16色表示対応）	基本グラフィックモード（８色表示対応）	プリンタポートの参照

（＊＊）BC1同期モード：送信タイミングには本体内タイマを使用，
受信タイミングにはモデムのクロックを使用．
ST2同期モード：送信，受信タイミングともにモデムのクロックを使用．
同期刻時機構：送信タイミングには本体内タイマを使用，
受信タイミングには受信データから作られるクロックを使用．
調歩同期モード：送信，受信タイミングともに本体内タイマを使用．

表1-4　ディップスイッチSW2の意味

スイッチ名	番号	目的	ON	OFF	ソフトからの参照手段
SW2	1				システムポートの参照
	2	ターミナルモードの指定	直接ターミナルモードを起動する	BASICモードにする	
	3	テキスト画面の表示文字数の指定	80文字／行	40文字／行	
	4		25行／画面	20行／画面	
	5	メモリスイッチ初期化の有無	メモリスイッチを変化させることができる	リセットするごとにメモリスイッチが初期化される	
	6	内蔵ハードディスクの切り離し指定	内蔵ハードディスクを切り離す	内蔵ハードディスクを使用する	
	7				
	8	GDC モードの選択	GDC 5MHzモード	GDC 2.5MHzモード	

表1-5　ディップスイッチSW3の意味

スイッチ名	番号	目的	ON	OFF	ソフトからの参照手段
SW3	1	内蔵FDの動作モードの指定	固定モード	自動切り替えモード	
	2		2DDモード	2HDモード	
	3				
	4				
	5				
	6	メインRAMの容量の指定	メインRAM 512KB	メインRAM 640KB	マウスポートの参照
	7				
	8	動作CPUの指定	動作CPU 80286/386	動作CPU V30	プリンタポートの参照

■1-6-2 メモリスイッチ

メモリスイッチの設定内容を記憶している不揮発性メモリは，ノーマルモードでは絶対メモリアドレスA3FE2Hから，ハイレゾモードではE3FE2Hからの領域に，4バイトおきに合計6バイト存在しています．その各ビットの意味を，図1-3～図1-8に示します．ディップスイッチと違って，メモリスイッチの各部分の意味付けは，機種による相違はほとんどありませんが，ハイレゾモードで有効なメモリスイッチは，MS-DOSで有効なもののみです．

メモリスイッチへのアクセスのしかたですが，読み出しについては普通のメモリに対してと同じ方

法でいつでも読み出すことができます．しかし，書き込みについては，メモリスイッチが誤って書きかえられることを防ぐために，普段は書き込み禁止状態になっているので，そのままでは書きかえることはできません．メモリスイッチを書きかえるときには，モードフリップフロップコントロール1（I/Oアドレス68H）に0DHを出力してメモリスイッチの書きかえを許可します．そして，書きかえが終わったら，同じモードフリップフロップ1に0CHを出力して再び書き込み禁止状態にしておきます．

図1-3　SW1（アドレスA3FE2H：ノーマル　E3FE2H：ハイレゾ）

図1-4　SW2（アドレスA3FE6H：ノーマル　E3FE6H：ハイレゾ）

ビット	値	内容
7	0	DELコード受信処理* 0：ターミナルモード(BS)　入出力モード(DEL) 1：ターミナルモード(NUL)　入出力モード(NUL)
6	0	テキスト画面の初期カラー指定　0：白　1：緑
5	0	数値演算プロセッサのクロック数　1：10MHz　2：8MHz/16MHz
4	0	数値演算プロセッサ(CPU：V30)　0：なし　1：あり
3	0	数値演算プロセッサ(CPU：8086/80286/80386(SX))　0：なし　1：あり
2	1	メモリサイズ　000：128キロバイト　011：512キロバイト 001：256キロバイト　100：640キロバイト 010：384キロバイト　100：768キロバイト （ハイレゾモードのみ）
1	0	
0	0	

0	4

16進数
（システム既定値）

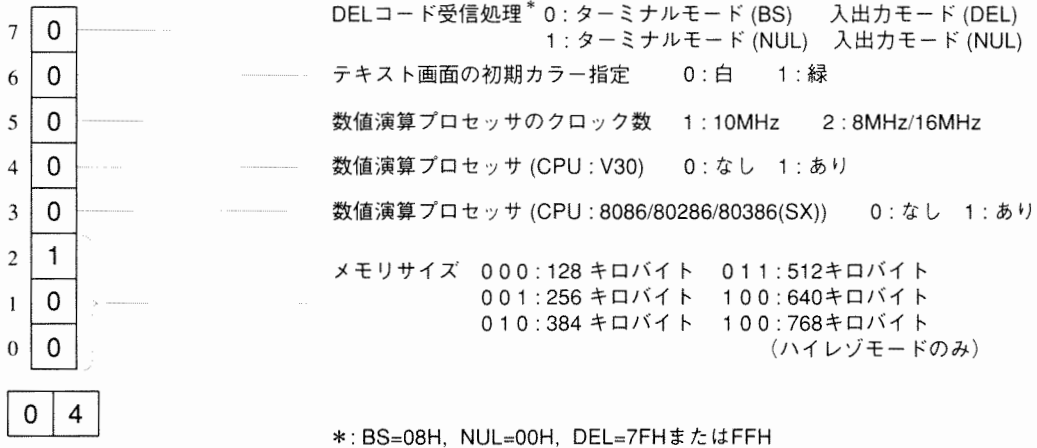
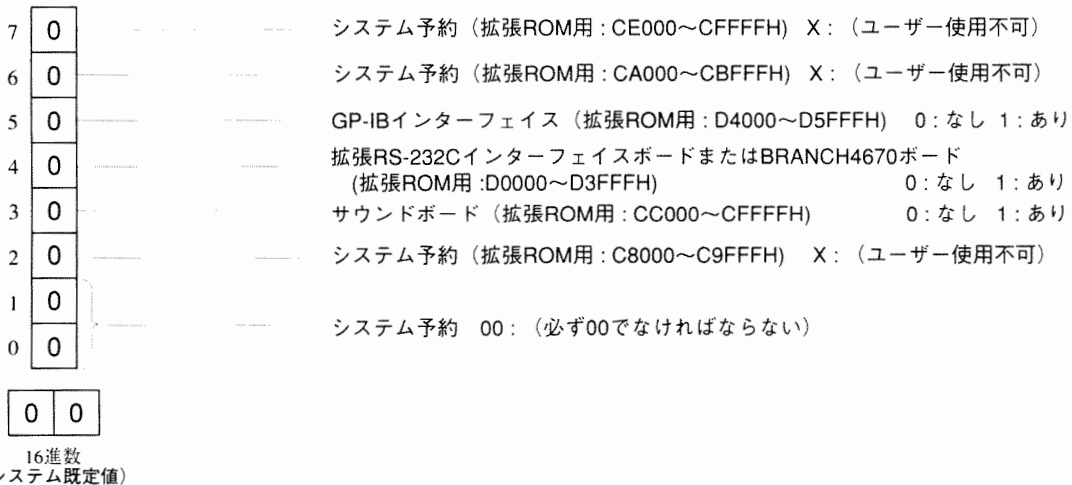
*：BS=08H，NUL=00H，DEL=7FHまたはFFH

ビット0, 1, 2, 3, 4, 6はN88-BASICとMS-DOSで機能する
ビット7はN88-BASICのみで機能する
ビット5はMS-DOSのみで機能する

図1-5　SW3（アドレスA3FEAH：ノーマル　E3FEAH：ハイレゾ）

16進数
（システム既定値）

ビット0, 1, 2, 4, 6, 7はN88-BASICとMS-DOSで機能する
ビット3および5はN88-BASICのみで機能する

図1-6　SW4（アドレスA3FEEH：ノーマル　E3FEEH：ハイレゾ）

ビット	値	内容
7	0	システム起動装置の指定 0000：フロッピーディスク→固定ディスクの順にサーチ 0010：640KBフロッピーディスクのみ 0100：1MBフロッピーディスクのみ 1010：固定ディスク#1のみ 1011：固定ディスク#2のみ 上記以外：ROMモードBASICが起動する
6	0	
5	0	
4	0	
3	0	画面ハードコピー　0：白黒画面ハードコピー*　1：カラー画面ハードコピー*
2	0	固定ディスクユーザー識別名　0：使用する　1：使用しない
1	0	固定ディスクデバイス名優先　0：使用しない(フロッピー→固定の順に割り振られ　1：使用する(固定→フロッピーの順に割り振られる
0	1	PC-PR201系プリンタ　0：使用しない　1：使用する

0 1

16進数
システム既定値)

*：PC-PR201V系プリンタが接続され、SW6のビット4=1のときのみ有効

ビット0, 1, 4, 5, 6, 7はN88-BASICとMS-DOSで機能する
ビット2, 3はN88-BASICのみで機能する

図1-7　SW5（アドレスA3FF2H：ノーマル　E3FF2H：ハイレゾ）

ビット	値	内容
7	0	未使用
6	0	
5	0	電話制御機能　0：使用しない　1：使用する
4	0	拡張機能制御機能（カラーハードコピー機能も可）　0：使用しない　1：使用する
3	0	モニターモード*　0：使用しない　1：使用する
2	0	
1	0	未使用
0	0	

*：PC-9801VF/VM0,2,4/UV21/VM21/VX0,2,4/UV21では、モニターモード拡張機能使用の有無となる

0 0

16進数
（システム既定値）

すべてのビットがN88-BASICのみで機能する

図1-8　SW6（アドレスA3FF6H：ノーマル）

第2部
98各機能の
標準的利用方法

§2-1 98各部の性能・機能

ここでは，98の各部分がどのような機能を持っていて，その性能はどうか，というようなことを述べます．これから各項に分かれて詳しく説明するハードの各部分の概略的な説明だと思ってください．それでは，98のハードの各部分の性能と機能について順に述べていきます．なお，ここに述べるのは各部分の主要な機能だけですので，詳しいことはそれぞれの項を参照してください．

●1 システムポート（☞2-2 システムポート，モードフリップフロップ）

システムポートは，システム情報の読み書きに使います．システムポートによって読み込むことができるシステム情報は，

- ・ディップスイッチの設定状況（一部のみ）
- ・RS-232Cの一部の信号
- ・ハードディスクのINT信号

等があります．システムポートへの出力でできるのは，

- ・RS-232C割り込みの設定
- ・ブザーのコントロール
- ・ソフトウェアリセットのモード設定
- ・一部プリンタ信号のコントロール

等です．

●2 タイマ（☞2-3 タイマ）

タイマは，システムクロックを基準にしてカウント動作を行います．その性能は，

- ・16ビットのカウント幅
- ・BCD（10進）またはバイナリー（2進）のカウント形式
- ・6モードのカウント方式（実際に使えるのは4モード）

等です．そしてその用途は，

- ・CPUに定期的に割り込みをかける
- ・RS-232Cにクロックを供給する
- ・ブザーの周波数を決定する

等です．

●3　カレンダー時計（☞ 2-4 カレンダー時計）

カレンダー時計は，電源を消しても動き続ける時計です．その用途は，

・現在の年月日時分秒を得る
・正確な 1 秒（0.5 秒）の時間間隔を得る

等です．

●4　キーボード（☞ 2-5 キーボード）

98 のキーボードの特徴としては，

・RS-232C と同じ形式で本体と通信
・キーボードコントローラによるキーリピート生成
・CAPS，カナキーロックのソフトウェアによる制御が可能

等があります．

●5　テキスト画面（☞ 2-6 テキスト）

テキストは，文字表示のためのものです．主な特徴は，

・8 × 16 ドットまたは 16 × 16 ドットの文字
・8 色の色指定
・リバース・ブリンク・シークレット等の特殊効果
・ユーザー定義文字
・スムーススクロール機能
・垂直同期信号による CPU への割り込み

等です．その用途は，

・文字や簡単なパターンを高速に表示する
・グラフィック画面を適当にマスクする

等です．

●6　テキスト GDC（☞ 2-6-3 テキスト GDC）

テキスト GDC は，テキスト画面を制御しています．その主な機能は，

・CRT の同期信号の生成
・テキスト画面の表示開始アドレスの決定
・カーソルの表示

などです．その用途は，

・カーソルの表示位置・形状などを制御する
・CRT の解像度を変化させる

などです.

●7　CRTC（☞2-6-4 CRTコントローラ）

CRTCは，GDCとともにテキスト画面を制御しています．その主な用途は，

- 文字の表示形態を制御する
- テキスト画面のスムーススクロールを実現する

などです.

●8　グラフィック（☞2-7 グラフィック）

グラフィックは，絵などの一般的な画像を表示するためのものです．98のグラフィック関連の機能としては，

- 640×200ドットあるいは640×400ドットの解像度
- モノクロ，8色中8色，4096色中16色の色指定
- 表示色を瞬時に変化させるパレット機能
- スクロール，縦方向拡大・縮小表示
- GDCによる線，円，拡大文字の描画
- グラフィックチャージャによる高速書き込み

等があります.

●9　メモリ（☞2-8 メモリ）

98には次のようなメモリがあります.

- メインメモリ
- EMS
- XMS

このうち最も高速で管理も簡単なのがメインメモリです.

●10　ディスク（☞2-9 ディスク）

98の標準的なディスクの特徴は，

- 2DD／2HDのディスクの読み書きが可能
- 2DDは約640KB，2HDは約1MBの記憶容量
- DMA転送による高速データ転送

等です．ディスク関係でできることは，

- ディスクに対してのデータの読み書き
- ディスクの記録方式等のチェック

等です．

●11　RS-232C（☞2-10 RS-232C）

RS-232C は，モデムなどの周辺機器や他のコンピュータとの通信を行うためのシリアルインターフェースです．その主な特徴は，

・データを1ビットずつやり取りする直列インターフェースである
・最大9600bps（1秒間に9600ビット）の通信速度
・システムクロックが5MHz系であれば19200bpsや38400bpsの通信速度が実現可能

等です．

●12　マウス（☞2-11 マウス）

マウス関係のハードでできることは，

・マウスの移動量とマウスボタンの状態の検出
・CPUに定期的に割り込みをかける
・一部のディップスイッチの状態検出

等です．

●13　プリンタ（☞2-12 プリンタ）

プリンタ関係でできることは，

・プリンタへの文字データとコントロールコードの出力
・プリンタの状況の検出
・一部のディップスイッチの状態検出

等です．

●14　FM音源（☞2-13 FM音源）

98の一般的なFM音源の特徴は，

・FM3音，SSG3音の同時発音が可能
・FM1音は，特殊な効果音のためのモードに設定できる

等です．FM音源関係のハードにできることは，

・FM音源による高度な発音，SSG音源による簡便な発音
・CPUに対して定期的に割り込みをかける
・ジョイスティックからの情報を読み取る

等です．

§ 2-2 システムポート

システムポートは，システム情報の参照と，システムの一部のコントロールをするための入力・出力ポートです．その目的のために，システムポートには8255AというLSIの相当品が使われています．8255Aは汎用の並列入出力LSIで，ポートA，ポートB，ポートCの3つの入出力ポートを持っていますが，システムポートではポートA，ポートBをシステム情報の入力に，ポートCをシステムの一部のコントロールに用いています．

システムポートが割り当てられているI/Oアドレスとそれぞれのポートの意味を表2-1に示します．

表2-1　システムポート関係のI/Oポート

リード ライト	I／O アドレス	機　　能	D7	D6	D5	D4	D3	D2	D1	D0
リード	31H	ポートAの読み出し （DIPSW2の設定状況）	$\overline{\text{SW8}}$	$\overline{\text{SW7}}$	$\overline{\text{SW6}}$	$\overline{\text{SW5}}$	$\overline{\text{SW4}}$	$\overline{\text{SW3}}$	$\overline{\text{SW2}}$	$\overline{\text{SW1}}$
	33H	ポートBの読み出し	$\overline{\text{CI}}$	$\overline{\text{CS}}$	$\overline{\text{CD}}$	INT3	CRTT	IMCK	EMCK	CDAT
	35H	ポートCの読み出し （診断用）	SHUT0	PSTBM	SHUT1	MCKEN	BUZ	TXRE	TXEE	RXRE
ライト	35H	ポートCの書き込み （一括書き込み）	SHUT0	PSTBM	SHUT1	MCKEN	BUZ	TXRE	TXEE	RXRE
	37H	ポートCの書き込み （個別書き込み）	0	0	0	0	ADR2	ADR1	ADR0	DT

ポートA（I/Oアドレス31H）には，ディップスイッチSW2の設定状態がそのまま反映されます．スイッチがONのときに対応するビットが0に，OFFのときに1になります．

ポートB（I/Oアドレス33H）には，いろいろな所からの情報が入力されますが，表2-1に示した各記号の意味は次の通りです．

ビット	信号名	意　　　味
2^7	$\overline{\text{CI}}$	ＲＳ－２３２ＣのＣＩ信号．呼び出されている状態であることを示すモデムからの信号です（９８ではサポートされません）．
2^6	$\overline{\text{CS}}$	ＲＳ－２３２ＣのＣＳ信号．モデムが送信可能であることを示す信号です．
2^5	$\overline{\text{CD}}$	ＲＳ－２３２ＣのＣＤ信号．モデムがキャリアを検出していることを示す信号です．
2^4	ＩＮＴ３	ハードディスクの割り込みが発生していることを示します．
2^3	ＣＲＴＴ	１のときに高解像度ディスプレイ（水平周波数24ＫＨｚ）が，０のときに標準解像度ディスプレイ（同15ＫＨｚ）が使われていることを示します．
2^2	ＩＭＣＫ	内蔵ＲＡＭにパリティエラーが発生したことを示します．
2^1	ＥＭＣＫ	拡張ＲＡＭにパリティエラーが発生したことを示します．
2^0	ＣＤＡＴ	カレンダ時計の直列データ出力がつながっています．

ポートC（I/Oアドレス35H，37H）は，ソフトウェアリセット後の動作等のコントロールに使います．表に示した各記号の意味は次の通りです．

ビット	信号名	意　　　味
2^7	ＳＨＵＴ０	ＳＨＵＴ１とともにソフトウェアリセット後の動作を制御する（表2-2参照）
2^6	ＰＳＴＢＭ	プリンタのＰＳＴＢ信号のマスク指定．
2^5	ＳＨＵＴ１	ＳＨＵＴ０とともにソフトウェアリセット後の動作を制御する（表2-2参照）
2^4	ＭＣＫＥＮ	メモリチェック結果の格納の制御．１のとき結果をＩＭＣＫ・ＥＭＣＫに格納する，０のとき結果を格納しない．
2^3	ＢＵＺ	ブザーの制御．１のときブザーＯＦＦ，０のときブザーＯＮ．
2^2	ＴＸＲＥ	ＲＳ－２３２Ｃの送信割り込み（ＴＸＲＤＹによる割り込み）の制御．１のとき送信割り込み許可，０のとき禁止．
2^1	ＴＸＥＥ	ＲＳ－２３２ＣのＴＸＥＭＰＴＹによる割り込みの制御．１のときＴＸＥＭＰＴＹ割り込み許可，０のとき禁止．
2^0	ＲＸＲＥ	ＲＳ－２３２Ｃの受信割り込み（ＲＸＲＤＹによる割り込み）の制御．１のとき受信割り込み許可，０のとき禁止．

表2-2　SHUT0とSHUT1の意味

SHUT0	SHUT1	ソフトウェアリセット後の動作
1	1	通常のリセットと同じ動作をする.
1	0	「SYSTEM SHUTDOWN」と表示して停止する.
0	×	CPUリセット後，プログラムの実行を継続する.

このポートCを書きかえるには，I/Oアドレス35Hに値を出力するのと，37Hに値を出力するという2通りの方法がありますが，ポートCの内容を一括して書きかえるときに35H，1ビットずつ書きかえるときに37Hを使います．37Hに値を出力したときにどこがどう変更されるかは表2-3の通りです．たとえば，C言語でI/Oアドレス37Hに値を出力してブザーをONにしたいときには，

```
outportb(0x37,0x06);
```

とすればよいのです．また，I/Oアドレス35Hからは現在の設定状況を読み出すことも可能です．35HからポートCを制御するときには通常，この設定状況を読み出し，変更したい部分のビットだけを書きかえて出力するようにします．

表2-3　I/Oポート37Hへ出力する値と動作の関係

ADR2	ADR1	ADR0	DT	37Hへの出力値	動　　作
0	0	0	0	00H	RXRE　禁止
0	0	0	1	01H	RXRE　許可
0	0	1	0	02H	TXEE　禁止
0	0	1	1	03H	TXEE　許可
0	1	0	0	04H	TXRE　禁止
0	1	0	1	05H	TXRE　許可
0	1	1	0	06H	BUZ ON
0	1	1	1	07H	BUZ OFF
1	0	0	0	08H	MCKEN 格納せず
1	0	0	1	09H	MCKEN 格納
1	0	1	0	0AH	SHUT1←0
1	0	1	1	0BH	SHUT1←1
1	1	0	0	0CH	PSTBM マスクせず
1	1	0	1	0DH	PSTBM マスク
1	1	1	0	0EH	SHUT0←0
1	1	1	1	0FH	SHUT0←1

§ 2-3 タイマ

98には，タイマLSIのμPD8253（または，相当品）が内蔵されています．このLSIは，16ビットカウンタを3つ内蔵しています（#0〜2）．値が設定されるとその値から，徐々に1ずつ引くという動作をします．98では，各カウンタが表2-4のように割り当てられています．また，動作モードは6種類あります．各モードは以下のような機能があります（98では，モード1，5はサポートされていません）．

①モード0（カウント終了時の割り込み）

モード指定後，OUTが0になり，カウント数をロードするとともに，カウントを開始し，カウントが終了すると，OUTが1となります．新しいカウント数をロードするまで，OUTは1となります．

②モード2（レートジェネレータ）

入力クロックに対してn分周を行い，デューティ比1/nの波形を出力します．（カウント終了時にのみ出力が1クロック分の1となり，そのほかのときは，0となるような波形になります）．

新たなカウント値が設定された場合は，次のサイクルから新しい値が使用されます．

③モード3（方形波ジェネレータ）

レートジェネレータと同様に，n分周しますが，こちらは，方形波を出力します．カウント数が偶数の場合のデューティ比は1/2，奇数の場合は，(n-1)/2nとなります．（0と1の割合が同じ波形になります）．

モード2同様，新たなカウント値が設定された場合は，次のサイクルから新しい値が使用されます．

④モード4（ソフトウエアトリガストローブ）

入力クロックに対してn分周を行い，デューティ比1／nの波形を出力します．（カウント終了時にのみ出力が1クロック分の間だけ1となり，その他の時は，0となるような波形になります）このモードでは，タイマLSIのGATE端子を操作することにより，カウンタの一旦停止，再開ができます．しかし，98では，この端子の操作はできません．したがって，モード2とまったく同じ動作になります．

新たなカウント値が設定された場合は，次のサイクルから新しい値が使用されます．

表2-4　カウンタの割り当て

カウンタ番号	割り当て
＃0	インターバルタイマ
＃1	スピーカー周波数設定 *
＃2	RS-232C通信速度設定

*PC-9801/E/F/Mではカウンタ#1はメモリリフレッシュに使用されています．スピーカー周波数の変更はできません．

■2-3-1 タイマのI/O

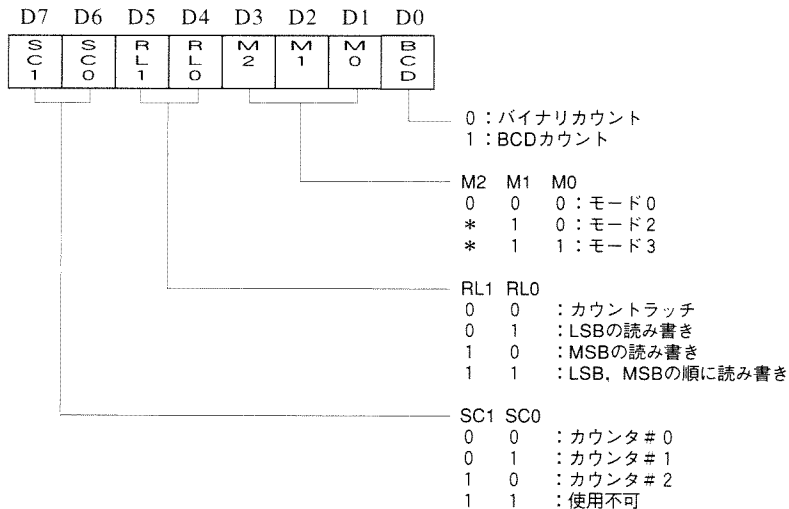
タイマのI/Oポート一覧を表2-4に示します．これらのI/Oポートを使うことによって，タイマを制御します．

表2-4 I/Oポート一覧

リード／ライト	ポートアドレス	命令	データ	説明
リード	71	READ #0	C7 ------------- C0 C15 ------------ C8	カウンタ0の値を読みだす
	73／3FDB*	READ #1	C7 ------------- C0 C15 ------------ C8	カウンタ1の値を読みだす
	75	READ #2	C7 ------------- C0 C15 ------------ C8	カウンタ2の値を読みだす
ライト	71	WRITE #0	C7 ------------- C0 C15 ------------ C8	カウンタ0に値を設定する
	73／3FDB*	WRITE #1	C7 ------------- C0 C15 ------------ C8	カウンタ1に値を設定する
	75	WRITE #2	C7 ------------- C0 C15 ------------ C8	カウンタ2に値を設定する
	77／3FDF**	MODE	S S R R M M M B C C L L 2 1 0 C 1 0 1 0 　　　D	各カウンタの動作モード設定

* PC-9801/E/F/Mでは，73H．そのほかの機種では，3FDBH
** PC-9801/E/F/Mでは，77H．そのほかの機種では，どちらでも可

◆MODE

各カウンタの動作モードを設定します．

◆WRITE　#0,1,2

カウンタに値を書き込みます．MODEでRL1,RL0=1,1を指定した場合は，16bitの値をLSB，MSBの順に2度書き込みを行います．どの書き込み方においても，値の書き込みが終了した時点で，カウントを開始します．

◆READ　#0,1,2

カウンタの値を読みだします．MODEでRL1,RL0=1,1を指定した場合は，2度読みだすことによって，16bitの値が，LSB，MSBの順に8bit単位で読みだされます．

●インターバルタイマ（カウンタ#0）の利用法

98では，インターバルタイマはカウンタ#0に割り当てられています．一定時間後に1度割り込みを発生させるのであればモード0，一定間隔でくり返し割り込みを発生させるのであればモード2またはモード3を使用します．

カウンタ#0に割り込み発生までの時間を設定（WRITE #0）します．カウンタに設定する値ですが，各機種のシステムクロック（CPUクロックではありません）により変わってきます．設定値をnとした場合のカウント時間を表2-6に示します．なお機種ごとのシステムクロックは，表2-7を参照してください．

タイマLSIの設定のほかに，割り込みコントローラの設定を行う必要があります．割り込みコントローラの設定は，「1-5．割り込み」を参照してください．

表2-6　システムクロックとカウント時間の関係

システムクロック	カウント時間
１０ＭＨｚ系	ｎ×１／２４５７．６ msec
８ＭＨｚ系	ｎ×１／１９９６．８ msec

表2-7　機種ごとのシステムクロック

システムクロック	主な機種
１０ＭＨｚ系	PC-9801RX/RA21/DA,全EPSON機
８ＭＨｚ系	PC-9801RA2/FA,H98

※98シリーズの場合，CPUクロックが8/16MHzのマシンでは，システムクロック=8MHz系が多く，CPUクロックが10/12/20MHzのマシンでは，システムクロック=10MHz系のマシンが多くなっています．

●スピーカー周波数設定（カウンタ#1）の方法

98では，スピーカー周波数の設定はカウンタ#1に割り当てられています．この場合，カウンタはモード3で使用します．

カウンタ#1にスピーカー周波数を決める値を設定（WRITE #1）します．カウンタに設定する値ですが，インターバルタイマの場合と同様に，各機種のシステムクロックにより変わってきます．設定値をnとした場合のスピーカー周波数を表2-8に示します．

なお，PC-9801/E/F/Mでは，この機能はサポートされていません．カウンタ#1は，メモリリフレッシュに使用されているため，カウンタの値を変更しないようにしてください．

表2-8　システムクロックとスピーカー周波数の関係

システムクロック	スピーカー周波数	nの初期値
１０ＭＨｚ系	２４５７．６／n(KHz)	１２２９
８ＭＨｚ系	１９９６．８／n(KHz)	９９８

●RS-232C通信速度設定（カウンタ#2）の方法

98では，RS-232Cの通信速度の設定は，カウンタ#2に割り当てられています．この場合，カウンタはモード2または，モード3で使用します．＊

カウンタ#2に通信速度を決める値を設定（WRITE #2）します．カウンタに設定する値ですが，インターバルタイマの場合と同様に，各機種のシステムクロックによって変わってきます．設定値と通信速度の関係を表2-9に示します．表中の1/16，1/64 モードは，RS-232CのシリアルI/O用LSIのモードです．詳しくは，「2-10.RS-232C」を参照してください．

＊どちらのモードでも通信速度の設定に使用できます．RS-232CのBIOSではモード3を利用しています．

表2-9　設定値と通信速度の関係

通信速度	1/16モード		1/64モード		1/1モード(同期式)	
(bps)	10MHz系	8MHz系	10MHz系	8MHz系	10MHz系	8MHz系
38400	4＊	------	------	------	64＊	52＊
19200	8＊	------	2＊	------	128＊	104＊
9600	16	13	4＊	------	256	208
4800	32	26	8	------	512	416
2400	64	52	16	13	1024	832
1200	128	104	32	26	2048	1664
600	256	208	64	52	4096	3328
300	512	416	128	104	8192	6656
150	1024	832	256	208	16384	13312
75	2048＊	1664*	512＊	416*	32768＊	26624＊

*印で示した値は，メーカーが保証する範囲（メーカー保証は9600bpsまで）を超えていますが，利用できる機種が多くあります．
また，最近の機種では19200bpsまでメーカーが保証しているものもあります．

■2-3-2 BIOSを使ったタイマの利用方法

●ノーマルモード

タイマーBIOSは，ノーマルモードにおいては，インターバルタイマの機能（カウンタ#0）のみのサポートと，少々おそまつなものとなっています．しかし，10ms単位でよい割り込み処理を行うには，手続きが簡単で容易に利用できます．また，タイマLSI直接制御と比べ，使用機種のシステムクロックの違いによる設定値の違いを考慮にいれなくてもよいという利点があります．

●ハイレゾモード

ハイレゾモードにおいては，ノーマルモードに比べ，インターバルタイマの機能が強化されています．一定間隔で割り込みを発生させたい場合でも，直接制御のほかにBIOSが利用できます．BIOSでは，マルチイベントタイマが利用できるという利点があります．これは，インターバルタイマに使えるカウンタは1つであるのに対し，BIOSで，ソフトウェア的に複数の割り込みルーチンを使い分けるものです．これにより，複数の異なった割り込み間隔のルーチンを呼び出すことができます．ただし，注意点としては，マルチイベント機能は，ソフトウェアで実現していることから，同時に複数のタイムアウトが発生すると，先の割り込みルーチン処理が終了するまで，後発の割り込みルーチン処理ができないというタイムラグが発生します．また，タイムアウトしたのは，1つの処理だけであっても，その処理時間分だけ，次の割り込み処理に入るまでの時間が遅くなります．

■タイマBIOS一覧（INT 1CH）

機能コード	機能	ノーマル	ハイレゾ
02H	インターバルタイマの設定	○	○
03H	タイマキャンセル	×	○
04H	インターバルタイマの設定（ワンショット）	×	○
05H	インターバルタイマの設定（リピーテッド）	×	○
06H	ビープ機能	×	○

1 インターバルタイマの設定

割り込み INT 1CH

入　力 AH←02H

CX←インターバルタイマ値（0～ffffH）

ES←呼び出し番地（セグメント）

BX←呼び出し番地（オフセット）

出　力 なし

解　説

CXにタイマの値を10ms単位で設定します．つまり，CX×10msとなります．設定した時間が経過した後に，実行中のプログラムを中断し（割り込みがかかり），ES:BXで示された番地がコールされます．

割り込みルーチンでは，使用するすべてのレジスタを保存する必要があることに注意してください．また，リターンにはiret命令を使用します．（一般にC言語には，このBIOSを簡単に利用する関数が用意されている場合がほとんどです．この関数を利用する場合は，このような細かな注意は必要ありません）．

サンプル

```
/*  プログラム実行から1秒後に t の値を2に変化させます。  */

#include  <stdio.h>
#include  <pc98.h>

int  t;

void  test(void)
{
        t  =  2;
}

void  main(void)
{
        t  =  1;
        pc98timer(100,  test);    /*  タイマBIOSの呼び出し  */
        while(1)  {
                printf("t  =  %d\n",  t);
        }
}
```

2 タイマキャンセル ハイレゾ

割り込み INT 1CH

入力 AH←03H

ES←キャンセルするパラメータブロック番地（セグメント）

BX←キャンセルするパラメータブロック番地（オフセット）

出力 AH→00H：正常終了

FFH：異常終了

解説 指定したパラメータブロックを削除します．つまり，インターバルタイマ設定（AH=05H,AH=06H）で設定した，待ちイベントの実行を中止します．パラメータブロックおよび，それに関する詳しい内容は，このタイマBIOSの項目の最後に示す「パラメータブロック」を参照してください．

3 インターバルタイマの設定（ワンショットモード） ハイレゾ

割り込み INT 1CH

入力 AH←04H

ES←パラメータブロックの番地（セグメント）

BX←パラメータブロックの番地（オフセット）

出力 AH→00H（常にこの値）

解説 ノーマル，ハイレゾ共通のタイマBIOSと同じ様に，指定した時間が経過した後に，1度だけ指定したルーチンが呼び出されます．このBIOSの場合，ノーマルモードと共通のBIOSと違い，マルチイベントが利用できます．各設定値はパラメータブロックに書き込みます．パラメータブロックおよび，それに関する詳しい内容は，「2-2-4　パラメータブロック」を参照してください．

割り込みルーチンにおいては，使用するすべてのレジスタを保存しなくてはいけません．また，割り込みルーチンから戻るときには，iret命令を使用します．

4 インターバルタイマの設定（リピーテッドモード）ハイレゾ

割り込み INT 1CH

入　力 AH←05H

ES←パラメータブロックの番地（セグメント）

BX←パラメータブロックの番地（オフセット）

出　力 AH→00H：処理ルーチンの呼び出し中止

FFH：処理ルーチンの呼び出し継続

解　説 与えられた間隔ごとに，指定した処理ルーチンを呼び出します．指定した処理ルーチンから戻るときに，次の割り込み処理を行うか指定できます．また，マルチイベントが利用できます．各設定値はパラメータブロックに書き込みます．パラメータブロックおよび，それに関する詳しい内容は，このタイマBIOSの項目の最後に示す「パラメータブロック」を参照してください．

割り込みルーチンにおいては，使用するすべてのレジスタを保存しなくてはいけません．また，割り込みルーチンから戻るときには，iret命令を使用します．

5 ビープ機能

割り込み INT 1CH

入　力 AH←06H

CX←ビープ時間（0001H～FFFFH（秒））

DX←周波数（0020H～8000H（Hz））

出　力 なし

解　説 指定した時間，指定した周波数で，ビープ音を鳴らします．CXに指定する値は秒単位，DXに指定する値はHz単位です．

■2-3-3 パラメータブロック

ハイレゾモードのタイマBIOSでは，マルチイベントの実現にパラメータブロックを使っています．パラメータブロックを1つの割り込み処理に対して，1つというように，割り込みごとに1つずつ用意します．そしてパラメータブロックの中のデータには次のパラメータブロックのアドレスも含み，パラメータブロックがつながっています．BIOSはタイマ値の小さいパラメータブロックから順番にならべ

ます（図2-1）．その際に「次のパラメータブロックのオフセット」，「次のパラメータブロックのセグメント」，「リピートワーク」をBIOSのワークエリアとして使用します．したがって，この3つはユーザーが設定する必要はありません．なお，次のパラメータブロックが存在しない場合は，次のパラメータブロックのオフセット，セグメントが0になります．

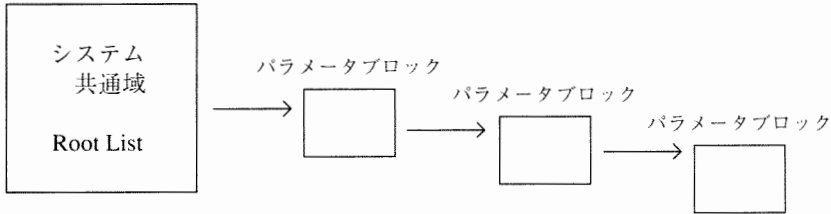



図2-1　パラメータブロックのつながり

1つのパラメータブロックのデータ形式は，図2-2のようになっています．タイマ値には，10ミリ秒単位で値を設定します．つまり，「設定した値×10ms」となります．

図2-2　パラメータブロックのデータ形式

§ 2-4 カレンダー時計

98には，カレンダー時計用LSIのμPD4990A（または相当品）が内蔵されています．このLSIは常にバッテリーでバックアップされています．これにより，年月日，曜日，時分秒を常に刻んでいます．このLSIを操作することによって，現在の日時の設定，読み出しを行うことができます．

（注：PC-9801／E／F／M／U2／VF2／VM0，1，2／UV2では，μPD1990AというLSIを使用しており，年の繰り上がり，閏年の自動判別はサポートされていません．）

カレンダー時計の制御には，一般にBIOSを利用します．旧機種とLSIが違うことや，直接制御をする必要性もさほどないことから，直接制御は一般的ではなく，やるべきではありません．

■2-4-1 ―――― カレンダー時計のI／O

カレンダー時計のI／Oポート一覧を表2-10に示します．

表2-10　カレンダー時計のI／Oポート一覧

リード ライト	I／Oポート	命令	データ	説明
ライト	20H	セットレジスタ	＊＊DCSCCC ILT210 KB	コマンドセット および データ書き込み
リード	33H	リードデータ	＊＊＊＊＊＊＊D O	時刻データ，1Hzの 信号の読み出し

●セットレジスタ

コマンドのセット，データの書き込みを行います．なお，シリアルコマンド表は表 2-11 に示します．

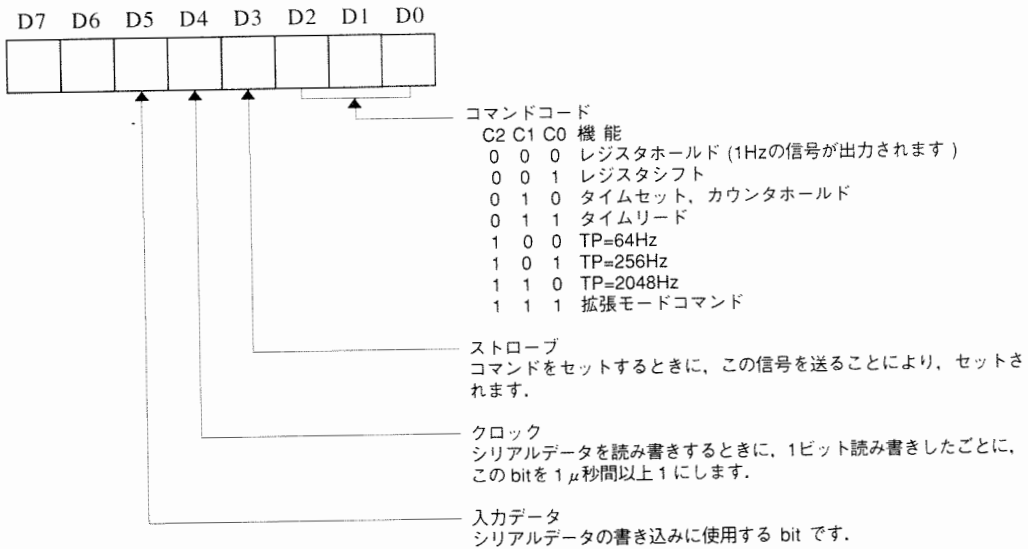
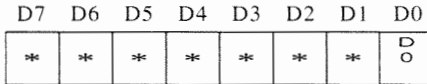



表 2-11　シリアルコマンド一覧

C3	C2	C1	C0	機能
0	0	0	0	レジスタホールド
0	0	0	1	レジスタシフト
0	0	1	0	タイムセット，カウンタホールド
0	0	1	1	タイムリード
0	1	0	0	TP = 64Hz
0	1	0	1	TP = 256Hz
0	1	1	0	TP = 2048Hz
0	1	1	1	TP = 4096Hz
1	0	0	0	TP = 1 秒　インタラプト出力／カウンタリセット
1	0	0	1	TP = 10 秒　〃
1	0	1	0	TP = 30 秒　〃
1	0	1	1	TP = 60 秒　〃
1	1	0	0	インタラプト出力リセット
1	1	0	1	インタラプトタイマスタート
1	1	1	0	インタラプトタイマストップ
1	1	1	1	テストモードセット

※シリアルコマンドは C0 → C1 → C2 → C3 の順に書き込みます．

●リードデータ

D7	D6	D5	D4	D3	D2	D1	D0
*	*	*	*	*	*	*	DO

出力データ
CLK を 1 μ秒間以上 1 にすると 図2-3 のようなデータが最下位 bit から順に出力されます．レジスタホールドのときは，ここから 1Hz の信号が出力されます．

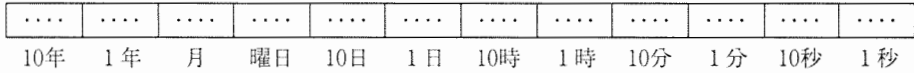
....	....	....	....	....	....	....	....	....	....	....	....
10年	1年	月	曜日	10日	1日	10時	1時	10分	1分	10秒	1秒

※データは，最下位 bit から入出力します．各値はそれぞれ，4bit からなり，合計で，全体が 48bit のデータとなります．

図 2-3　入出力データ形式

■2-4-2　カレンダー時計の BIOS

カレンダー時計の LSI の旧機種との違い，シリアルコマンドというものがあり，直接制御を行うためには，プログラムが少々複雑になるということからも，カレンダー時計の制御には，これから紹介する BIOS を利用するのが一般的であり，便利であると思われます．

■カレンダー時計 BIOS 一覧（INT 1CH）

機能コード	機能	ノーマル	ハイレゾ
00H	日付，時刻の読み出し	○	○
01H	日付，時刻の設定	○	○

■入出力データフォーマット

カレンダー時計 BIOS で使用する入出力データは全部で 6 バイトで構成され，図 2-4 のようなフォーマットになっています．また，各項目のデータ形式は，表 2-12 のようになってます．

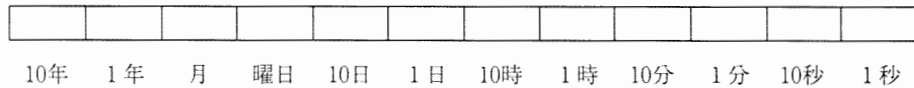



図 2-4　入出力データの形式

表 2-12　各項目のデータ形式

項目	データ形式	データ範囲
10年	BCD	0～9H
1年	BCD	0～9H
月	16進	1～CH
曜日	16進	0～6H
10日	BCD	0～3H
1日	BCD	0～9H
10時	BCD	0～2H
1時	BCD	0～9H
10分	BCD	0～5H
1分	BCD	0～9H
10秒	BCD	0～5H
1秒	BCD	0～9H

1　日付，時刻の読み出し

割り込み INT　1CH

入　　力 AH←00H

ES　←日付，時刻データを書き込む番地（セグメント）

BX　←日付，時刻データを書き込む番地（オフセット）

（データフォーマットに関しては58ページを参照）

出　　力 なし

解　　説 現在の日付，時刻を読み出し，ES：BX で示された番地に書き込みます．

サンプル /* 現在の日付，時刻データを読み出して表示します．　for TC++ BC++ */

```
#include <stdio.h>
#include <dos.h>

struct SREGS segregs;
union REGS inregs, outregs;

void main(void)
{
      int i;
      char date_data[7];

      segread(&segregs);
      segregs.es = segregs.ds;          /* date_dataのセグメントをesに入
                                        れます */
      inregs.x.bx = (int)date_data;     /* date_dataのオフセットをbxに入
```

```
                                                  れます */
        inregs.h.ah = 0;                          /* ahに0を入れます */
        int86x(0x1c, &inregs, &outregs, &segregs);    /* BIOSの呼び
                                                      出し */
        for(i = 0; i <= 5; i++) {
              printf("%02x ", (date_data[i] & 0xff));
        }
}
```

2 日付，時刻の設定

割り込み INT 1CH

入　力 AH ← 01H

ES ←設定する日付，時刻データがある番地（セグメント）

BX ←設定する日付，時刻データがある番地（オフセット）

（データフォーマットに関しては58ページを参照）

出　力 なし

解　説 ES：BXで示された番地のデータに基づき，日付，時刻を設定します．

§2-5 キーボード

PC98シリーズのキーボードには，さまざまな種類のキーボードがありますが，ここでは，主に図2-5のキーボードについて説明します．

キーボードの主な働きは，どのキーが押されたか，または，どのキーが離されたか，という情報を本体側に伝えることです．このデータの形式は図2-6のようになっています．

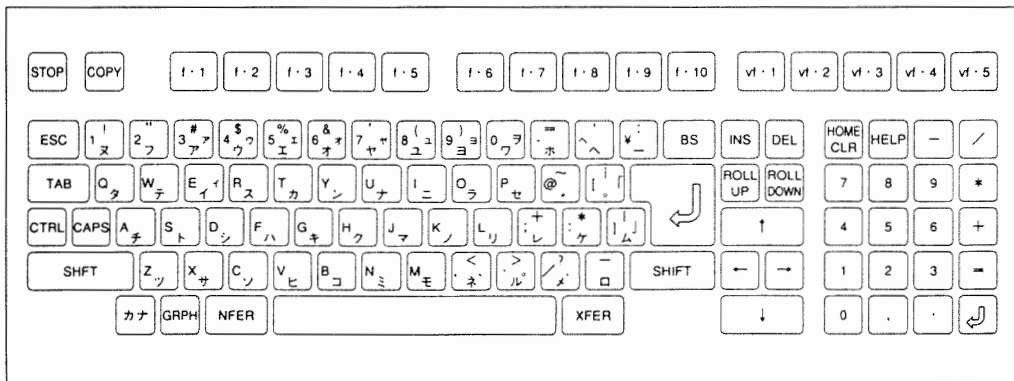



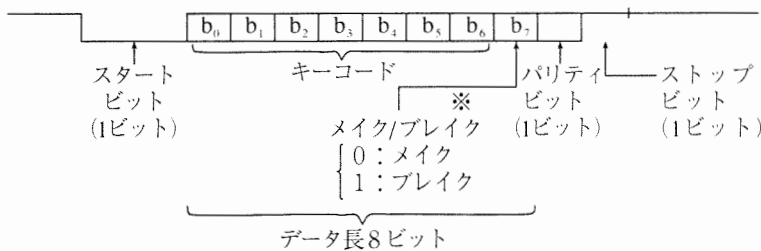
図2-5　キーボード

※メイク　：キーが押されたときの割り込みを示す．
※ブレイク：キーが離されたときの割り込みを示す．

図2-6　キーボードのデータ形式

キーコードは，表2-13に示してあるように，各キー1つ1つに一対一に対応している7ビットのデータです．キーボードから本体に送られるデータは，このキーコードにキーが押されたか，離されたかの状態を示す1ビットの情報を加えたものです．これをスキャンコードと呼びます．

しかし，キーボードのANKキー（英数字かなキー）は，シフトキーと同時に押すことによりキャラ

クタを選択するようになっていますので，キーコードだけでは，どのキャラクタが選択されているか，判断することができません．したがって，本体側でシフトキーの状態を保持して，キーコードからキャラクタと一対一に対応しているキーデータを作り出します．また，キーコードとキーデータとを合わせたものをキーコードデータと呼び表2-14のようになっています．

表2-13　キーコード表

上位3ビット→　／　下位4ビット↓

	0	1	2	3	4	5	6	7
0	ESC	Qﾀ	Fﾊ	,<ﾈ	-	.	STOP	SHFT
1	1!ﾇ	Wﾃ	Gｷ	.>ﾙ	/	NFER	COPY	CAPS
2	2"ﾌ	Eｲ	Hｸ	/?ﾒ	7	VF1	F1	ｶﾅ
3	3#ｱ	Rｽ	Jﾏ	_ﾛ	8	VF2	F2	GRPH
4	4$ｳ	Tｶ	Kﾉ	空白	9	VF3	F3	CTRL
5	5%ｴ	Yﾝ	Lﾘ	XFER	*	VF4	F4	
6	6&ｵ	Uﾅ	;+ﾚ	RLUP	4	VF5	F5	
7	7'ﾔ	Iﾆ	:*ｹ	RLDN	5		F6	
8	8(ﾕ	Oﾗ	]}ﾑ	INS	6		F7	
9	9)ﾖ	Pｾ	Zﾂ	DEL	+		F8	
A	0ﾜ	@ﾞ	Xｻ	↑	1		F9	
B	-=ﾎ	[{ﾟ	Cｿ	←	2		F10	
C	^ﾍ	複改	Vﾋ	→	3			
D	¥¦ｰ	Aﾁ	Bｺ	↓	=			
E	BS	Sﾄ	Nﾐ	HMCR	0			
F	TAB	Dｼ	Mﾓ	HELP	,			

キーボード上のキーひとつひとつにつけられた通し番号がコード．
たとえば，［G］キーのキーコードは21H．
復改はリターンキー，空白はスペースキー．
フルキーの上段に並んでいる数字キーとテンキー上の数字キーとは区別できるが，
2個のリターンキー，シフトキーは区別できない．

表2-14 キーコードデータ表(1)

キーコード		通常		SHIFT		CAPS		CAPS + SHIFT		カナ		カナ + SHIFT		GRPH		CTRL	
00	ESC	00	1B	00	1B	00	1B	00	1B	00	1B	00	1B	00	1B	00	16
01	! 1 ヌ	01	31	01	21	01	31	01	21	01	C7	01	C7				
02	" 2 フ	02	32	02	22	02	32	02	22	02	CC	02	CC				
03	# 3 ァ ア	03	33	03	23	03	33	03	23	03	B1	03	A7				
04	$ 4 ゥ ウ	04	34	04	24	04	34	04	24	04	B3	04	A9				
05	% 5 ェ エ	05	35	05	25	05	35	05	25	05	B4	05	AA	05	F2		
06	& 6 ォ オ	06	36	06	26	06	36	06	26	06	B5	06	AB	06	F3		
07	' 7 ャ ヤ	07	37	07	27	07	37	07	27	07	D4	07	AC	07	F4		
08	(8 ュ ユ	08	38	08	28	08	38	08	28	08	D5	08	AD	08	F5		
09	) 9 ョ ヨ	09	39	09	29	09	39	09	29	09	D6	09	AE	09	F6		
0A	0 ヲ ワ	0A	30	0A	30	0A	30	0A	30	0A	DC	0A	A6	0A	F7		
0B	= - ホ	0B	2D	0B	3D	0B	2D	0B	3D	0B	CE	0B	CE	0B	8C		
0C	` ^ ヘ	0C	5E	0C	60	0C	5E	0C	60	0C	CD	0C	CD	0C	8B	0C	1E
0D	¦ ¥ ー	0D	5C	0D	7C	0D	5C	0D	7C	0D	B0	0D	B0	0D	F1	0D	1C
0E	BS	0E	08	0E	08	0E	08	0E	08	0E	08	0E	08	0E	08	0E	08
0F	TAB	0F	09	0F	09	0F	09	0F	09	0F	09	0F	09	0F	09	0F	09

キーコード		通常		SHIFT		CAPS		CAPS + SHIFT		カナ		カナ + SHIFT		GRPH		CTRL	
10	Q タ	10	71	10	51	10	51	10	71	10	C0	10	C0	10	9C	10	11
11	W テ	11	77	11	57	11	57	11	77	11	C3	11	C3	11	9D	11	17
12	E ィ イ	12	65	12	45	12	45	12	65	12	B2	12	A8	12	E4	12	05
13	R ス	13	72	13	52	13	52	13	72	13	BD	13	BD	13	E5	13	12
14	T カ	14	74	14	54	14	54	14	74	14	B6	14	B6	14	EE	14	14
15	Y ン	15	79	15	59	15	59	15	79	15	DD	15	DD	15	EF	15	19
16	U ナ	16	75	16	55	16	55	16	75	16	C5	16	C5	16	F0	16	15
17	I ニ	17	69	17	49	17	49	17	69	17	C6	17	C6	17	E8	17	09
18	O ラ	18	6F	18	4F	18	4F	18	6F	18	D7	18	C7	18	E9	18	0F
19	P セ	19	70	19	50	19	50	19	70	19	BE	19	BE	19	8D	19	10
1A	‾ @ ゛	1A	40	1A	7E	1A	40	1A	7E	1A	DE	1A	DE	1A	8A	1A	00
1B	{ [゜ 「	1B	5B	1B	7B	1B	5B	1B	7B	1B	DF	1B	A2			1B	1B
1C	↲	1C	0D	1C	0D	1C	0D	1C	0D	1C	0D	1C	0D	1C	0D	1C	0D
1D	A チ	1D	61	1D	41	1D	41	1D	61	1D	C1	1D	C1	1D	9E	1D	01
1E	S ト	1E	73	1E	53	1E	53	1E	73	1E	C4	1E	C4	1E	9F	1E	13
1F	D シ	1F	64	1F	44	1F	44	1F	64	1F	BC	1F	BC	1F	E6	1F	04

左：キーコード　右：キーデータ　2つあわせて：キーコードデータ

表2-14　キーコードデータ表(2)

キーコード		通常		SHIFT		CAPS		CAPS + SHIFT		カナ		カナ + SHIFT		GRPH		CTRL	
20	F ハ	20	66	20	46	20	46	20	66	20	CA	20	CA	20	E7	20	06
21	G キ	21	67	21	47	21	47	21	67	21	B7	21	B7	21	EC	21	07
22	H ク	22	68	22	48	22	48	22	68	22	B8	22	B8	22	ED	22	08
23	J マ	23	6A	23	4A	23	4A	23	6A	23	CF	23	CF	23	EA	23	0A
24	K ノ	24	6B	24	4B	24	4B	24	6B	24	C9	24	C9	24	EB	24	0B
25	L リ	25	6C	25	4C	25	4C	25	6C	25	D8	25	D8	25	8E	25	0C
26	; + レ	26	3B	26	2B	26	3B	26	2B	26	DA	26	DA	26	89		
27	: * ケ	27	3A	27	2A	27	3A	27	2A	27	B9	27	B9	27	94		
28	] } 」 ム	28	5D	28	7D	28	5D	28	7D	28	D1	28	A3			28	1D
29	Z ツ	29	7A	29	5A	29	5A	29	7A	29	C2	29	AF	29	80	29	1A
2A	X サ	2A	78	2A	58	2A	58	2A	78	2A	BB	2A	BB	2A	81	2A	18
2B	C ソ	2B	63	2B	43	2B	43	2B	63	2B	BF	2B	BF	2B	82	2B	03
2C	V ヒ	2C	76	2C	56	2C	56	2C	76	2C	CB	2C	CB	2C	83	2C	16
2D	B コ	2D	62	2D	42	2D	42	2D	62	2D	BA	2D	BA	2D	84	2D	02
2E	N ミ	2E	6E	2E	4E	2E	4E	2E	6E	2E	DO	2E	DO	2E	85	2E	0E
2F	M モ	2F	6D	2F	4D	2F	4D	2F	6D	2F	D3	2F	D3	2F	86	2F	0D

キーコード		通常		SHIFT		CAPS		CAPS + SHIFT		カナ		カナ + SHIFT		GRPH		CTRL	
30	, < 、 ネ	30	2C	30	3C	30	2C	30	3C	30	C8	30	A4	30	87		
31	. > 。 ル	31	2E	31	3E	31	2E	31	3E	31	D9	31	A1	31	88		
32	/ ? ・ メ	32	2F	32	3F	32	2F	32	3F	32	D2	32	A5	32	97		
33	‾ ロ			33	5F			33	5F	33	D8	33	D8			33	1F
34	SPACE	34	20	34	20	34	20	34	20	34	20	34	20	34	20	34	20
35	XFER	35	00	A5	00	35	00	A5	00	35	00	A5	00	35	00	B5	00
36	ROLL UP	36	00	36	00	36	00	36	00	36	00	36	00	36	00	36	00
37	ROLL DOWN	37	00	37	00	37	00	37	00	37	00	37	00	37	00	37	00
38	INS	38	00	38	00	38	00	38	00	38	00	38	00	38	00	38	00
39	DEL	39	00	39	00	39	00	39	00	39	00	39	00	39	00	39	00
3A	↑	3A	00	3A	00	3A	00	3A	00	3A	00	3A	00	3A	00	3A	00
3B	←	3B	00	3B	00	3B	00	3B	00	3B	00	3B	00	3B	00	3B	00
3C	→	3C	00	3C	00	3C	00	3C	00	3C	00	3C	00	3C	00	3C	00
3D	↓	3D	00	3D	00	3D	00	3D	00	3D	00	3D	00	3D	00	3D	00
3E	HOME CLR	3E	00	AE	00	3E	00	AE	00	3E	00	AE	00				
3F	HELP	3F	00	3F	00	3F	00	3F	00	3F	00	3F	00	3F	00	3F	00

表2-14　キーコードデータ表(3)

キーコード		通常		SHIFT		CAPS		CAPS + SHIFT		カナ		カナ + SHIFT		GRPH		CTRL	
40	-	40	2D	40	2D	40	2D	40	2D	40	2D	40	2D	40	2D	40	2D
41	/	41	2F	41	2F	41	2F	41	2F	41	2F	41	2F	41	2F	41	2F
42	7	42	37	42	37	42	37	42	37	42	37	42	37	42	98	42	37
43	8	43	38	43	38	43	38	43	38	43	38	43	38	43	91	43	38
44	9	44	39	44	39	44	39	44	39	44	39	44	39	44	99	44	39
45	*	45	2A	45	2A	45	2A	45	2A	45	2A	45	2A	45	95	45	2A
46	4	46	34	46	34	46	34	46	34	46	34	46	34	46	E1	46	34
47	5	47	35	47	35	47	35	47	35	47	35	47	35	47	E2	47	35
48	6	48	36	48	36	48	36	48	36	48	36	48	36	48	E3	48	36
49	+	49	2B	49	2B	49	2B	49	2B	49	2B	49	2B	49	R0	49	2B
4A	1	4A	31	4A	31	4A	31	4A	31	4A	31	4A	31	4A	93	4A	31
4B	2	4B	32	4B	32	4B	32	4B	32	4B	32	4B	32	4B	8F	4B	32
4C	3	4C	33	4C	33	4C	33	4C	33	4C	33	4C	33	4C	92	4C	33
4D	=	4D	3D	4D	3D	4D	3D	4D	3D	4D	3D	4D	3D	4D	96	4D	3D
4E	0	4E	30	4E	30	4E	30	4E	30	4E	30	4E	30	4E	9A	4E	30
4F	,	4F	2C	4F	2C	4F	2C	4F	2C	4F	2C	4F	2C	4F	90	4F	2C

キーコード		通常		SHIFT		CAPS		CAPS + SHIFT		カナ		カナ + SHIFT		GRPH		CTRL	
50	.	50	2E	50	2E	50	2E	50	2E	50	2E	50	2E	50	9E	50	2E
51	NFER	51	00	A1	00	51	00	A1	00	51	00	A1	00	51	00	B1	00
52	vf1	52	00	C2	00	52	00	C2	00	52	00	C2	00	D2	00		
53	vf2	53	00	C3	00	53	00	C3	00	53	00	C3	00	D3	00		
54	vf3	54	00	C4	00	54	00	C4	00	54	00	C4	00	D4	00		
55	vf4	55	00	C5	00	55	00	C5	00	55	00	C5	00	D5	00		
56	vf5	56	00	C6	00	56	00	C6	00	56	00	C6	00	D6	00		

表2-14　キーコードデータ表(4)

キーコード		通常		SHIFT		CAPS		CAPS+SHIFT		カナ		カナ+SHIFT		GRPH	CTRL	
60	STOP															
61	COPY															
62	f1	62	00	82	00	62	00	82	00	62	00	82	00		92	00
63	f2	63	00	83	00	63	00	83	00	63	00	83	00		93	00
64	f3	64	00	84	00	64	00	84	00	64	00	84	00		94	00
65	f3	65	00	85	00	65	00	85	00	65	00	85	00		95	00
66	f3	66	00	86	00	66	00	86	00	66	00	86	00		96	00
67	f3	67	00	87	00	67	00	87	00	67	00	87	00		97	00
68	f3	68	00	88	00	68	00	88	00	68	00	88	00		98	00
69	f3	69	00	89	00	69	00	89	00	69	00	89	00		99	00
6A	f3	6A	00	8A	00	6A	00	8A	00	6A	00	8A	00		9A	00
6B	f3	6B	00	8B	00	6B	00	8B	00	6B	00	8B	00		9B	00

キーコード		通常	SHIFT	CAPS	CAPS+SHIFT	カナ	カナ+SHIFT	GRPH	CTRL
70	SHIFT								
71	CAPS								
72	カナ								
73	GRAPH								
74	CTRL								

■2-5-1 キーボードのI／Oポート

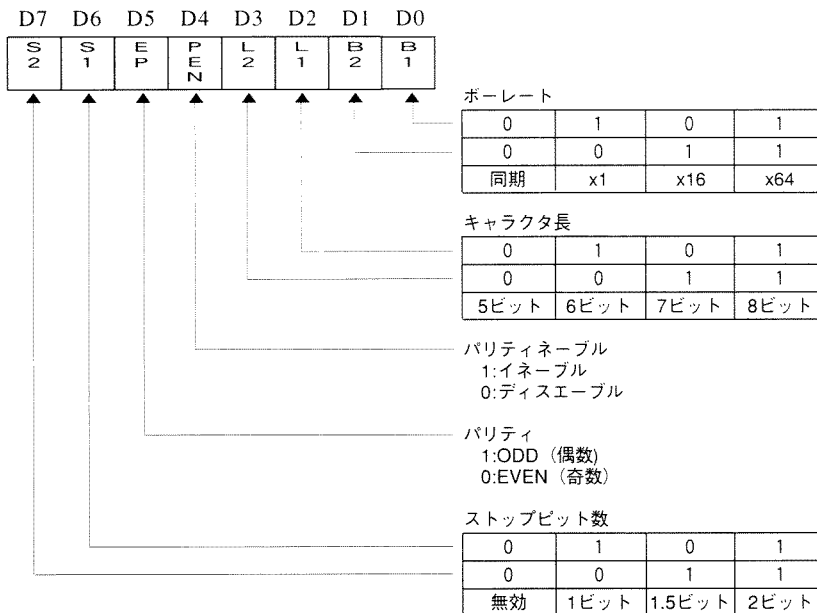
キーボードから送られてきたデータは，本体側のμPD8251Aが受け取ります．したがって，μPD8251Aを直接制御することによって，キーボードからのデータを読み取ることができますが，BIOSを使って安全に，しかも簡単にできるので，普通はBIOSを使って行います．

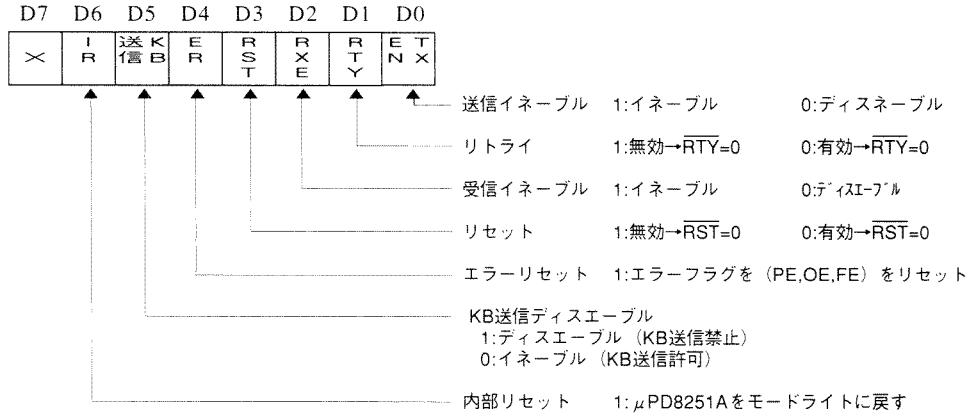
キーボードのI／Oポート一覧を表2-15に示します．個々の命令については，以下に説明します．

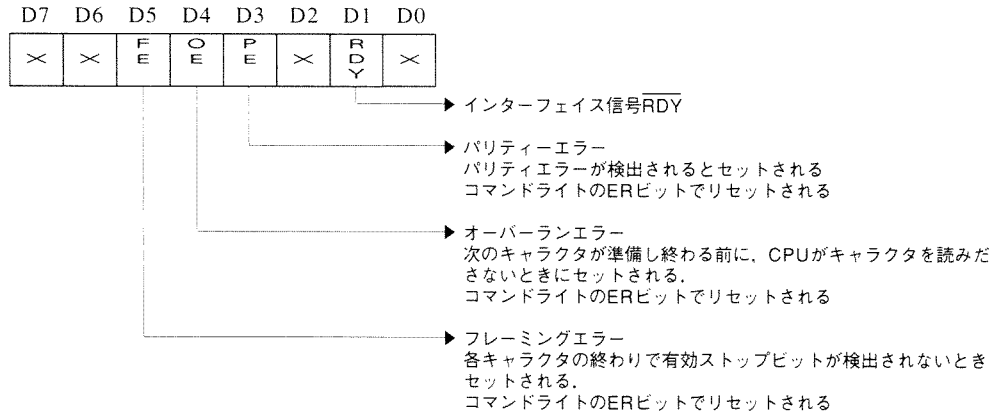
表2-15　キーボードのI／Oポート

命令	I/Oポートアドレス	R/W	機能
モードライト	43H	W	モードセット
コマンドライト	43H	W	コマンドセット
データリード	41H	R	μPD8251にロードされた1バイトのデータを読む．
ステータスリード	43H	R	μPD8251のステータスを読む

◆モードライト

μPD8251Aの初期化を行います．ただし，μPD8251Aの内部または外部のリセット動作の後に実行する必要があります．

◆コマンドライト

μPD8251Aの動作を指示します．この命令はモードライト命令実行後に，受け付け可能になります．ただし，一度モードライト命令が行われると，これ以降はすべてコマンドライトとして受け取られます．

◆ステータスリード

μPD8251Aのステータス情報を読み込みます．

■2-5-2 キーボードBIOS

キーボードBIOSはどのようにキー入力の処理を行っているのでしょうか．仮に，私たちが作るプログラムが，INT18Hの内部割り込みによって，キーボードBIOSを呼び出したときだけキー入力の有無を調べるようになっていたとしたら，プログラムが別の処理をしている間はキー入力を受け付けなくなってしまいます．

したがって，キー入力があって，キーボードからデータが送られてきたら，ハードウェア割り込み09HによってキーボードBIOSは呼び出され，システム共通域にあるキーバッファにキー入力の状態を格納するようになっています．この処理によって，プログラムが別の処理をしている間に入力されたキーの処理ができるようになっています．もちろん，リアルタイムに現在どのキーが押されているかを調べることもできます．では，キーバッファがシステム共通域にどのように存在しているがを表2-16に示しておきます．

表2-16　キーバッファ

番地	容量	解　　説
0000:0502	32バイト	2バイトのキーコードデータのバッファで16文字が格納できるリングバッファ
0000:0522	2バイト	キーコードを変換するテーブルのオフセットアドレス
0000:0524	2バイト	キーコードデータのバッファの先頭を示すヘッドポインタ
0000:0526	2バイト	キーコードデータのバッファの空の先頭を示すテイルポインタ
0000:0528	1バイト	バッファに格納されているキーコードデータの個数を示すバッファカウンタ
0000:0529	1バイト	エラーが発生しリトライした回数
0000:052A	1バイト	キーボード入力状態テーブル
0000:053A	1バイト	シフトキー状態バッファ

表2-16からわかるように，キーバッファは16文字分までしか格納することができません．それ以上入力されたキーコードデータは，切り捨てられ，ビープ音を発生させます．このビープ音は，表2-17のようにして制御できます．

表2-17　キーボードバッファのオーバーフロー

0000:0500HのBIT5	1	ビープ音を鳴らす
	0	ビープ音を鳴らさない

またSTOPキーの場合は06H，COPYキーの場合は05Hの内部割り込みを発生させます．次にINT18Hで呼び出されるキーボードBIOS示します．

■キーボードBIOS一覧（INT18H）

機能コード	機能	ノーマル	ハイレゾ
00H	キーデータの読み出し	○	○
01H	キーバッファ状態の取得	○	○
02H	シフトキー状態の検査	○	○
03H	キーボードインターフェースの初期化	○	○
04H	キー入力状態の検査	○	○
05H	キーバッファからのキーコードの読み出し	○	○
06H	バッファの初期化	×	○
07H	シフトキー状態とキーデータの読み出し	×	○
08H	シフトキー状態とキーデータの検査	×	○
09H	キーデータの作成	×	○

■ハイレゾの場合

基本的にINT18Hで，キーボードBIOSを呼び出したときに返ってくる値はノーマルの場合と同じですが内部処理の方法が異なります．システム共通領域も表2-18のように拡張されています．

表2-18　ハイレゾのシステム共通領域

番地	容量	解説
0000:0408	8バイト	シフトキーコード
0000:040F	4バイト	キーコードのバッファアドレス（上位2バイトがオフセット下位2バイトがセグメント）
0000:0418	4バイト	内部割り込みテーブルアドレス
0000:0522	1バイト	キーコードデータのバッファのサイズ

シフトキーはユーザーによって定義できるようになっています．つまり，シフトキーコードに格納されているキーコードがシフトキーとみなされます．順序はシフトキー状態バッファに対応しています．

バッファに格納されるデータは，キーコードデータではなくてキーコードとそのときのシフトキーの状態であります．また，バッファのアドレスと容量はユーザーで指定できます．

内部割り込みを発生させるキーもユーザーが定義することができます．まず，割り込みテーブルの1バイト目に割り込みを発生させるキーの個数を格納し，次からそのキーコードを格納します．そのとき，キーコードの最上位ビットが1のときはINT05Hを0のときはINT06Hを発生させます．

表2-19　シフトキーコード（8バイト）の初期値

番地	キーコード	キー
0000:0408H	0FFH	
0000:0409H	0FFH	
0000:040AH	0FFH	
0000:040BH	74H	CTRL
0000:040CH	73H	GRAPH
0000:040DH	72H	カナ
0000:040EH	71H	CAPS
0000:040FH	70H	SHIFT

表2-20　割り込みキーボードテーブルの初期値

個数	02H		
キーコード	60H	STOPキー	INT06H
キーコード	0E1H	COPYキー	INT05H

1 キーデータの読み出し

割り込み INT18H

入　力 AH←00H

出　力 AX←キーコードデータ

（AH←キーコード，AL←キーデータ：表2-14参照）

解　説 キーコードバッファの先頭のキーコードデータを読み出し，バッファの先頭を示すヘッドポインタを移動させ，次の読み込みに備えます．キーコードバッファにデータが格納されていない場合は，格納されるまで待ちます．ただし，SHIFTキーとSTOPキーは内部割り込みを発生させバッファに格納されないので，読み出すことはできません．また，このBIOSをコールする前に，キーボードインターフェースの初期化を行ってください．

サンプル

```
/*　キーデータを表示させる．*/
#include<dos.h>
#include<stdio.h>
void main()
{
        union REGS in, out;
        in.h.ah=0x03;                /* キーボードインターフェースの初期化 */
        int86(0x18, &in, &out);
```

```
        printf("何かキーを押してください．\n");
        in.h.ah=0x00;
        int86(0x18, &in, &out);
        printf("キーコードデータ=%x\n",out);
}
```

2 キーバッファ状態の取得

割り込み INT18H

入　力 AH←01H

出　力 AX←キーコードデータ

（AH←キーコード，AL←キーデータ：表2-14参照）

DH←01H：AXに読みだしたデータ有効

00H：AXに読みだしたデータ無効

解　説 キーコードバッファの先頭に格納されているキーコードデータを読み出します．このときに，「キーデータの読み出し」（AH←0）とは異なりバッファの先頭を示すヘッドポインタは移動させません．したがって，このBIOSコールによってバッファの状態は変化しません．また，データが格納されているかをBXレジスタに出力するので，入力待ちになることはありません．

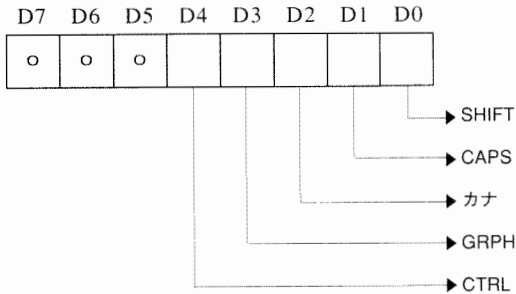
サンプル

```
/*　キーバッファ状態を表示させる．*/
#include<dos.h>
#include<stdio.h>
void main()
{
        union REGS in, out;
        in.h.ah=0x03;                /*　キーボードインターフェースの初期化　*/
        int86(0x18, &in, &out);
        printf("何かキーを押してください．\n");

        in.h.ah=0x01;
        int86(0x18, &in, &out);
        while(out.h.bh==0){
                int86(0x18, &in, &out);
                if (out.h.bh==1) printf("キーコードデータÅÅ%x\n",out);
                if (out.h.bh==0) printf("キー入力なし\n");
        }
}
```

3 シフトキー状態の取得

割り込み INT18H

入　力 AH←02H

出　力 AL←シフトキーの状態

解　説 シフトキーが現在押されているかを調べて，その状態をALレジスタに格納します．キーが押されているときは，そのキーの対応するビットが1になります．押されていないときは，そのキーの対応するビットが0になります．

サンプル

```
/*  シフトキーの状態を表示させる. */
#include<dos.h>
#include<stdio.h>
void main()
{
        union REGS in, out;

        in.h.ah=0x03;                /*   キーボードインターフェースの初期化   */
        int86(0x18, &in, &out);

        printf("何かキーを押してください. \n");
        in.h.ah=0x01;
        int86(0x18, &in, &out);
        while(out.h.bh==0) int86(0x18, &in, &out);

        in.h.ah=0x02;
        int86(0x18, &in, &out);
        printf("シフトキーの状態\n");
                printf("SHIFTキー  %x\n",out.h.al&0x01);
                printf("CAPSキー   %x\n",(out.h.al&0x02)>>1);
                printf("カナキー   %x\n",(out.h.al&0x04)>>2);
                printf("GRPHキー   %x\n",(out.h.al&0x08)>>3);
                printf("CAPSキー   %x\n",(out.h.al&0x10)>>4);
}
```

4　キーボードインターフェースの初期化

出　力　INT18H

解　説　AH←03H

割り込み　なし

入　力

キーボードインターフェースを初期化して，システム共通域のキーバッファをクリアします．キーボードBIOSを呼び出すときは，まずこのBIOSを呼び出してください．

サンプル

キーデータの読み出し（p.71）などのほかのキーボードBIOSのサンプルを参照してください．

5　キー入力状態の取得

割り込み　INT18H

入　力　AH←04H

AL←キーコードグループの番号

出　力　AH←入力で指定されたキーコードグループの8キーの状態

解　説

ALレジスタで指定したキーコードグループの各キーの状態を調べてAHレジスタに格納します．このBIOSではキーバッファによって状態を調べていないので，キーバッファに対して，影響をあたえません．

キーコードグループ＼ビット	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	ESC	! 1ヌ	" 2フ	#ァ 3ア	$ゥ 4ウ	%ェ 5エ	&ォ 6オ	'ャ 7ヤ
1	(ュ 8ユ	)ョ 9ヨ	ヲ 0ワ	＝ ーホ	^ヘ	¦ ¥ー	BS	TAB
2	Qタ	Wテ	ィ Eイ	Rス	Tカ	Yン	Uナ	Iニ
3	Oラ	Pセ	~ @゛	{「 [゜	↵	Aチ	Sト	Dシ
4	Fハ	Gキ	Hク	Jマ	Kノ	Lリ	+ ;レ	* :ケ
5	}」]ム	ッ Zツ	Xサ	Cソ	Vヒ	Bコ	Nミ	Mモ
6	<、 ,ネ	>。 .ル	?・ /メ	_ □	SPACE	XFER	ROLL UP	ROLL DOWN
7	INS	DEL	↑	←	→	↓	HOME CLR	HELP
8	－	／	7	8	9	*	4	5
9	6	＋	1	2	3	＝	0	,
A	・	NFER	vf・1	vf・2	vf・3	vf・4	vf・5	
B							HOME	
C	STOP	COPY	f・1	f・2	f・3	f・4	f・5	f・6
D	f・7	f・8	f・9	f・10				
E	SHIFT	CAPS	カナ	GRPH	CTRL			
F								

サンプル

```
/*    キー入力状態の取得     */
#include<dos.h>
#include<stdio.h>
void main()
{
      int i;
      union REGS in, out;

      in.h.ah=0x03;                /*  キーボードインターフェースの初期化  */
      int86(0x18, &in, &out);

      printf("何かキーを押してください. \n");
      in.h.ah=0x01;
      int86(0x18, &in, &out);
      while(out.h.bh==0) int86(0x18, &in, &out);

      for(i=0;i<=0x0f;i++){
            in.h.ah=0x04;
            in.h.al=i;
            int86(0x18, &in, &out);
            printf("キーコードグループ%x = %x %x %x %x %x %x %x %x\n",
                      i,out.h.ah&0x01,(out.h.ah&0x02)>>1,
                      (out.h.ah&0x04)>>2,(out.h.ah&0x08)>>3,
                      (out.h.ah&0x10)>>4,(out.h.ah&0x20)>>5,
                      (out.h.ah&0x40)>>6,(out.h.ah&0x80)>>7);
      }
}
```

6 キーバッファからのキーコードの取得

割り込み **INT18H**

入　力 **AH←05H**

出　力 **AH**←キーコード

AL←キーデータ

BX←01H：AXに読み出したデータ有効

00H：AXに読み出したデータ無効

解　説　キーコードバッファの先頭に格納されているキーコードデータを読み出します．このとき，バッファの先頭を示すヘットポインタは移動させません．したがって，このBIOSコールによってバッファの状態は変化しません．また，データが格納されているかをBXレジスタに出力するので，入力待ちになることはありません．

サンプル

```
/*  キーコードとキーデータを表示させる. */
#include<dos.h>
#include<stdio.h>
void main()
{
```

```
        union REGS in, out;
        in.h.ah=0x03;                    /*  キーボードインターフェースの初期化  */
        int86(0x18, &in, &out);
        printf("何かキーを押してください. ");

        in.h.ah=0x05;
        int86(0x18, &in, &out);
        while(out.h.bh==0){
                int86(0x18, &in, &out);
                if (out.h.bh==1) {
                        printf("キーコード=%x\n",out.h.ah);
                        printf("キーデータ=%x\n",out.h.al);
                }
                if (out.h.bh==0) printf("キー入力なし\n");
        }
}
```

7 バッファの初期化

割り込み INT18H

入　力 AH←06H

出　力 なし

解　説 キーコードバッファを初期化します.

8 シフトキー状態とキーコードの取得

割り込み INT18H

入　力 AH←07H

出　力 AH←キーコード
AL←キーデータ
BL←シフトキー状態

解　説 　キーコードバッファの先頭に格納されているキーコードのキーデータとシフトキー状態を読みだします. キーコードバッファにデータが格納されていない場合は, 格納されるまで待ちます.

サンプル

```
/*  キーデータを表示させる. */
#include<dos.h>
#include<stdio.h>
void main()
{
```

```
        union  REGS  in,  out;
        in.h.ah=0x06;                    /*   キーボードバッファの初期化   */
        int86(0x18,  &in,  &out);
        printf("何かキーを押してください. ");

        in.h.ah=0x07;
        int86(0x18,  &in,  &out);
        printf("キーコード=%x\n",out.h.ah);
        printf("キーデータ=%x\n",out.h.al);
        printf("シフトキーの状態\n");
        printf("SHIFTキー  %x\n",out.h.bl&0x01);
        printf("CAPSキー   %x\n",(out.h.bl&0x02)>>1);
        printf("カナキー    %x\n",(out.h.bl&0x04)>>2);
        printf("GRPHキー   %x\n",(out.h.bl&0x08)>>3);
        printf("CAPSキー   %x\n",(out.h.bl&0x10)>>4);
}
```

9 シフトキー状態とキーコードの検査 ハイレゾ

割り込み INT18H

入　　力 AH←08H

出　　力 AH←キーコード
AL←キーデータ
BL←シフトキー状態

解　　説 キーコードバッファの先頭に格納されているキーコードのキーデータとシフトキー状態を読みだします．このとき，キーコードバッファの状態を調べるだけで，バッファの先頭を示すヘットポインタは移動しません．

サンプル

```
/*  キーデータを表示させる. */
#include<dos.h>
#include<stdio.h>
void main()
{
        int  keycord;
        union  REGS  in,  out;
        in.h.ah=0x03;                    /*   キーボードバッファの初期化   */
        int86(0x18,  &in,  &out);
        printf("何かキーを押してください. ");

        in.h.ah=0x08;
        int86(0x18,  &in,  &out);
        keycord=out.h.ah;
              while(out.h.ah==keycord){
              in.h.ah=0x08;
              int86(0x18,  &in,  &out);
        }
printf("キーコード=%x\n",out.h.ah);
```

```
        printf("キーデータ=%x\n",out.h.al);
        printf("シフトキーの状態\n");
        printf("SHIFTキー  %x\n",out.h.bl&0x01);
        printf("CAPSキー    %x\n",(out.h.bl&0x02)>>1);
        printf("カナキー    %x\n",(out.h.bl&0x04)>>2);
        printf("GRPHキー    %x\n",(out.h.bl&0x08)>>3);
        printf("CAPSキー    %x\n",(out.h.bl&0x10)>>4);
}
```

10 キーデータの作成

割り込み INT18H

入　力 AH←09H
AL←キーコード
BL←シフトキー状態

出　力 AH←キーコード
AL←キーデータ

解　説 キーコードとシフトキー状態よりキーデータを作成します．．

サンプル

```
/*  キーデータの作成する  */
#include<dos.h>
#include<stdio.h>
void main()
{
        union REGS in, out;
        printf("キーコードを入力してください．");
        scanf("%x",&in.h.al);
        printf("シフトキー状態を入力してください．");
        scanf("%x",&in.h.bl);
        in.h.ah=0x09;
        int86(0x18, &in, &out);
        printf("キーデータ=%x",out.h.al);
}
```

§2-6 テキスト

98の画面表示方式には，テキストとグラフィックの2つの方式があり，それらが合成されて98の画面を構成しています．そのうち，ここで説明するテキストは，文字表示を専門に扱っている画面です．つまり，ある程度決まった文字（パターン）しか表示できないかわりに，短い文字コードを指定するだけで文字の表示ができる画面なのです．98のテキスト画面は，数バイトを書き込むだけで半角文字から全角漢字までを，色や特殊効果を付けて表示することが可能です．この強力なテキスト画面を持っていることが，98の文字表示が他機種よりも高速である大きな要因になっています．

テキスト画面は，画面表示関係のハードウェアによって，後述するグラフィック画面と重ね合わされてから画面上に表示されるのですが，その際には，常にテキスト画面の方が優先して表示されます．つまり，テキスト画面の表示があると，その部分のグラフィック画面の表示はその後に隠れてしまう，ということです．98では，この優先順位を変更することはできません．しかし，この特性を利用すると，グラフィック画面の表示内容をテキスト画面の表示で適当に隠してやることで，グラフィック画面の表示を面白い方法で出現させる，などということも可能になってきます．

98のテキストに関わるハードウェアとしては，テキストVRAM，テキストGDC，CRTC，CG，モードフリップフロップがあります．テキストVRAMは表示する文字とその色や属性の決定に，テキストGDC，CRTC，モードフリップフロップは表示形式の決定やCRTのコントロールに，CGは文字パターンの管理に用いられています．以下では，これらについて順に説明していきます．

■2-6-1 テキストVRAM

テキストVRAMは，テキスト画面に表示する文字のキャラクタコードや色等を指定するためのRAM領域で，ノーマルモードでは絶対メモリアドレスA0000H～A3FFFH，ハイレゾモードでは同じくE0000H～E3FFFHに存在しており，その形式は図2-21のようになっています．図に示した通り，98のテキストVRAM（以下，TVRAMと略）には文字エリアとアトリビュートエリアの2種類があります．文字エリアには表示する文字のキャラクタコード（文字コード）を，アトリビュートエリアには，文字の色や反転などの特殊効果についての情報を書き込みます．文字エリアでは画面上の1桁について2バイトが割り当てられていて，1文字表示するのに最低2バイト，全角漢字を表示するときには4バイト（=2桁分）を書き込む必要があります．また，98では通常，1行は80桁ですから，80×2=160バイトで1行となります．それに対し，アトリビュートエリアでは偶数番地のみにRAMが存在し，1桁について1バイトのアトリビュート情報を書き込むことができます．

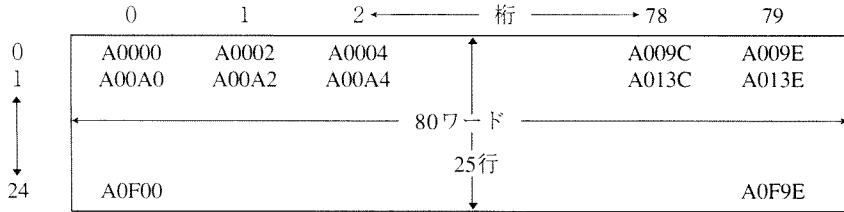
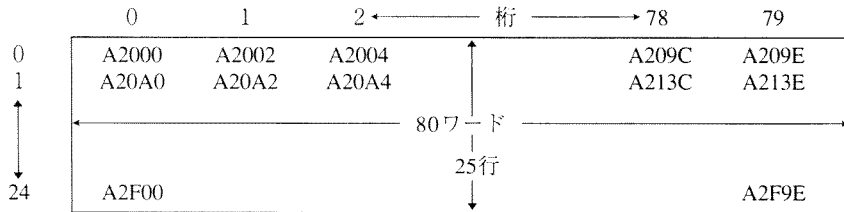
●文字エリア

●アトリビュートエリア（偶数番地だけを使用した2Kバイト）

＊ノーマルモードの場合．ハイレゾモードの場合は，メモリアドレスの最上位の「A」を「E」に置きかえて考えてください．

図2-21　テキストVRAMの形式

●文字エリア

TVRAMにキャラクタコードを書き込むことによって表示できる文字は，大きく分けると，ANK文字，全角漢字，半角漢字の3種類です．ANK文字は，種類が英数字・カナなどに限られている文字で，1桁で1文字表示することができ，キャラクタコードは1バイトです．全角漢字は，横幅がANK文字の倍で，2桁で1文字表示することができ，キャラクタコードは2バイトです．ユーザー定義文字などもこの部類に入ります．半角漢字は，ANK文字と同じ横幅で，1桁に1文字表示できますが，キャラクタコードは2バイトです．

これら3種類の文字を，画面上の任意の位置に表示したい場合に，それぞれTVRAMのどの部分にどのようなコードを書き込めばよいかを，以下に列挙してみます．

1）ANK文字の場合

まず，コードを書き込むべきアドレスを考えます．その文字を表示させたい画面上の位置を（X,Y）とすると，1行が160バイト，1桁が2バイトですから，コードを書き込み始める先頭アドレスをADRとすると，

ADR=（文字エリアの先頭アドレス）+Y×160+X×2

となります．文字エリアの先頭アドレスは，通常はA0000Hです．ANK文字は1桁で1文字で，1桁は2バイトですから，ADRのアドレスから2バイトのコードを書き込む必要があります．このうち，ADRのアドレスにはANK文字のキャラクタコードを，ADR＋1のアドレスには，00Hを書き込みます（図2-22参照）．

たとえば，C言語で，（40,12）の位置に'A'の字を表示したいときには，'A'の字のキャラクタコード

は41Hなので,

```
poke(0xa000,12 * 160 + 40 * 2, 0x0041);
```

とします.

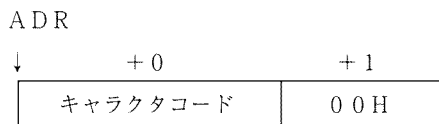



図2-22　TVRAM上のANK文字の形式

2）全角漢字の場合

全角漢字の場合も，コードを書き込み始めるべき先頭アドレスをADRとすると，ANK文字のときと同じように，

```
ADR=(文字エリアの先頭アドレス)+Y×160+X×2
```

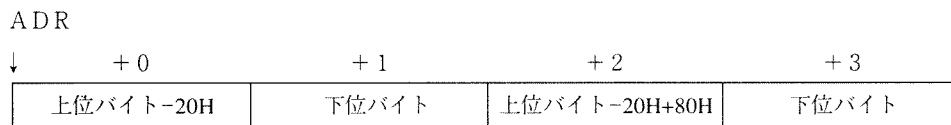
となります．全角漢字は1文字で2桁のスペースを取りますから，書き込むべきコードは4バイトです．この4バイトにどのようなコードを書き込めばよいかは，図2-23を参照してください．

上位バイト：JISコードの上位バイト．JISコード3021Hの漢字なら30H.
下位バイト：JISコードの下位バイト．JISコード3021Hの漢字なら21H.
80x86CPUとは上位・下位の順番が逆であることに注意.

図2-23　TVRAM上の全角漢字の形式

たとえば，C言語で，（20,10）の位置に'技'の字を表示したいときには，'技'の字のJISコードは353BHなので，

```
poke (0xa000, 10 * 160 + 20 * 2, 0x3B35 - 0x20);
poke (0xa000, 10 * 160 + 21 * 2, 0x3B35 - 0x20 + 0x80);
```

とします．コードの上位と下位が逆になっていることに注意してください．

なお，図中のJISコードというのは，通常のJISコードのことで，MS-DOSで用いられているシフトJISコードではないので注意してください．MS-DOSで用いられている漢字を表示するときには，コード変換が必要になります．また，この全角漢字表示で書き込むべき4バイトのうち，前2バイトの部分と後2バイトの部分とには通常，同じ文字の文字コードを書き込みますが，これらに異なる文字の文字コードを書き込んだ場合，前2バイトがJIS第1，第2水準漢字，JIS非漢字ならば，後2バイトは無視されます

が，前2バイトがそれ以外の漢字（罫線文字，特殊記号など）やユーザー定義文字の場合には，前2バイトの漢字が左半分だけ表示され，後2バイトの文字がその隣に表示されます．このことを利用すると，本来全角文字であるユーザー定義文字を，あたかも半角文字のように扱うことも可能になります．

3）半角漢字の場合

半角漢字の場合も，コードを書き込む先頭アドレスは上の2つと同じです．半角漢字は，1桁で1文字表示できますから，書き込むべきコードは2バイトです．その2バイトに書き込むべきコードは，図2-24に示した通りです．

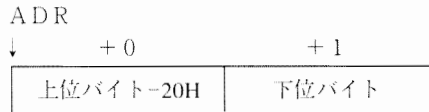



図2-24　TVRAM上の半角漢字の形式

たとえば，C言語で，（40,12）の位置に半角漢字の'A 'の字を表示したいときには，'A 'の字のキャラクタコードは2941Hなので，

```
poke (0xa000, 12 * 160 + 40 * 2, 0x4129 - 0x20);
```

とします．

●アトリビュートエリア

次に，アトリビュートエリアに値を書き込むことによって文字に色をつけたり，反転などの特殊効果を出したりする方法について述べます．

表示文字の色や特殊効果を指定したいときは，その文字のキャラクタコードを格納している文字エリアと同じ場所に相当するアトリビュートエリアにアトリビュートデータを書き込みます．具体的には，（X,Y）の位置に表示する文字の属性を指定したいときには，

（アトリビュートエリアの先頭アドレス）＋Y×160＋X×2

で計算されるアドレスに1バイトのアトリビュートデータをセットします（アトリビュートエリアの先頭アドレスは通常A2000H）．このとき，書き込むアトリビュートデータの形式を図2-25に示します．この図より，たとえば通常の特殊効果なしの白い文字を表示したいときにはE1Hを，水色の反転文字を表示したいときにはA5Hを書き込めばよいことがわかります．

このようにして，TVRAMに文字のキャラクタコードとアトリビュートコードを書き込んでおくと，CRTCが定期的にTVRAMの内容を読み込み，そこに書き込まれているキャラクタコードの文字の文字パターンをCGから読み出して，その文字にアトリビュートコードで指定された属性を付けてディスプレイに表示してくれます．

なお，このテキストVRAMにアクセスするときには，次のようなことに注意する必要があります．

1）テキストVRAMは，アクセススピードが非常に遅い

2）テキストVRAMは，たとえ32ビットマシンでも16ビットバスを介してつながっている

G	R	B	表示色
0	0	0	黒
0	0	1	青
0	1	0	赤
0	1	1	紫
1	0	0	緑
1	0	1	水色
1	1	0	黄色
1	1	1	白

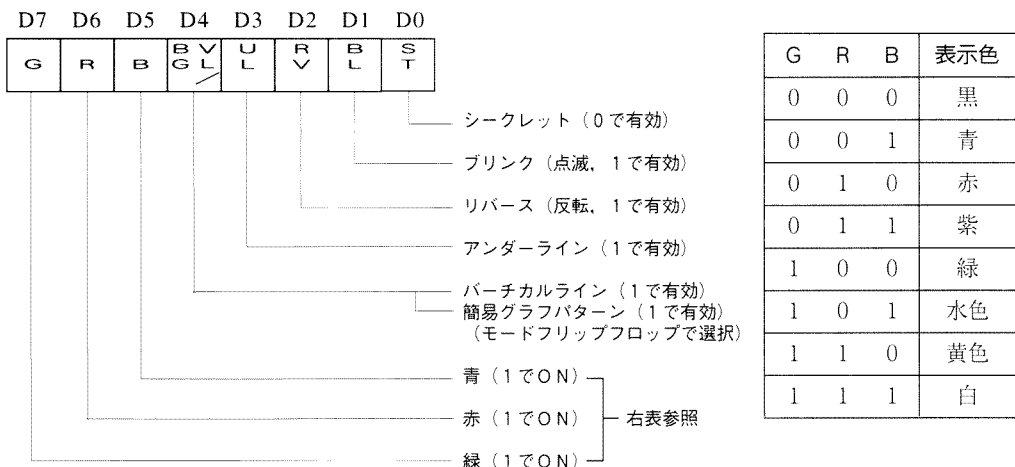



図2-25　アトリビュートデータの形式

1）は，主にハードウェアによるウェイトのためと思われます．機種によっても異なりますが，テキストVRAMにアクセスするのに必要な時間（アクセスタイム）は，メインRAMのアクセスタイムの3～12倍もかかってしまいます．特に，性能の高い機種で隔差が大きい傾向にあります．したがって，効率のよいプログラムを組むためには，できるかぎり偶数番地からのワードアクセスを心がけることはもちろん，スクロールなどさせるときはメインRAM上に仮想VRAMを設けてやる，などというのも有効になります．

2）は，ただでさえ遅いテキストVRAMをさらに遅くする要因になっているものです．32ビットマシンでは，テキストVRAMは遅いのですから32ビット（4バイト）一括転送をしたくなりますが，物理的なバス幅が16ビットであるために，32ビットアクセスをすると転送が最低でも2度起こるので，16ビットアクセスと比べて速くなることは期待できません．したがって，32ビットマシンだからと無理に32ビット転送をする必要はありません．

何にしろ，この2つは98の画面表示が遅い大きな要因の１つになっているので，ハードメーカーには早急に改善してほしいと思います．

■サンプルプログラム

画面上にいろいろな形式や色をした文字を表示します．

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>

void main(void)
{
  clrscr();
  poke       (0xa000,  6*160+20*2,0x0041);    /*  'A'(ｺｰﾄﾞ41H)を(20,6)の位置に表示 */
  pokeb      (0xa200,  6*160+20*2,0xe1);      /*  'A'は白の通常表示  */
  poke       (0xa000,  6*160+21*2,0x00b1);    /*  'ｱ'(ｺｰﾄﾞB1H)を(21,6)の位置に表示 */
```

```
    pokeb     (0xa200,  6*160+21*2,0xa5);    /* 'ア'は水色の反転表示  */
    poke      (0xa000,  8*160+20*2,0x2130-0x20);    /*'亜'(JISコード3021H)の左側*/
    poke      (0xa000,  8*160+21*2,0x2130-0x20+0x80);/*'亜'(JISコード3021H)の右側*/
    pokeb     (0xa200,  8*160+20*2,0xc7);    /* '亜'の左側は黄色の反転・点滅  */
    pokeb     (0xa200,  8*160+21*2,0xc3);    /* '亜'の右側は黄色の点滅  */
    poke      (0xa000,10*160+20*2,0x4129-0x20); /* 'Ａ'(JISコード2941H)を表示 */
    pokeb     (0xa200,10*160+20*2,0x89);     /* 'Ａ'は緑色でアンダーライン付き  */
}
```

■2-6-2 テキストのI/Oポート

テキスト関係のハードウェアを制御するI/Oポートは，I/Oアドレス60H～6EHの偶数番地に割り当てられています．それぞれのI/Oアドレスに何がつながっているかを，表2-22に示します．

表2-22　テキスト関係のI/Oポート

リード／ライト	I／O アドレス	機　　能	データ D7 D6 D5 D4 D3 D2 D1 D0
リード	60H	GDCステータスの読み出し	←GDCステータスフラグ→
	62H	GDCデータの読み出し	←GDCデータ→
ライト	60H	GDCパラメータの書き込み	←GDCパラメータ→
	62H	GDCコマンドの書き込み	←GDCコマンド→
	64H	CRT割り込みリセット	任意の値
	68H	モードフリップフロップ1のコントロール	0 0 0 0 MFA2 MFA1 MFA0 MFDT
	6CH	ボーダーカラーの書き込み	I* G R B 0 0 0 0

＊H98の16色モード時のみ有効

表の中に出てくる用語のうち，GDCの詳細については，「2-6-3.テキストGDC」を参照してください．

CRT割り込みリセットというのは，CRTの垂直同期信号が発生したときにPICに割り込み要求が出されるようにするためのものです．ふだん何もしなければ，CRTの垂直同期信号が発生しても，PICに割り込み要求は出されませんが，このCRT割り込みリセットのポートに何か値を出力すると（どんな値でもよい），その後に発生した最初の垂直同期信号のときにかぎり，PICにCRTV割り込みの割り込み要求が出されます．したがって，垂直同期信号が発生したら毎回，割り込みがかかるようにしたいときには，CRTV割り込みがかかるたびにこのCRT割り込みリセットを発行してやる必要があります．

たとえば，C言語でCRT割り込みリセットをするときは，

```
outportb(0x64,  0x00);
```

とします．

モードフリップフロップ（以下モードF/Fと略）というのは，システムの動作モードを指定するため

の1ビットだけのメモリの集まり，というようなものです．モードF/F1では，表中のMFA0～MFA2で書きかえるフリップフロップを，MFDTでどう書きかえるかを指定します．具体的な出力する値と設定内容の関係については表2-23を参照してください．たとえば，C言語で，モードF/F1をコントロールして，ANKを6×8ドットにしたいときには，

```
outportb(0x68, 0x06);
```

とし，これを7×13に戻したいときには，

```
outportb(0x68, 0x07);
```

とします．

表2-23 I/Oアドレス68Hに出力する値と設定内容の関係

ＭＭＭＭ ＦＦＦＦ ＡＡＡＤ ２１０Ｔ	６８Ｈに 出力する 値	意　　味	解　　説
０ ０ ０ ０	００Ｈ	ＡＴＲ４がバーチカルライン	テキストアトリビュートのビット４の意味の選択
０ ０ ０ １	０１Ｈ	ＡＴＲ４が簡易グラフ	
０ ０ １ ０	０２Ｈ	カラーグラフィックモード	グラフィック画面のカラーモードの選択
０ ０ １ １	０３Ｈ	モノクログラフィックモード	
０ １ ０ ０	０４Ｈ	テキスト８０字モード	テキスト画面の表示桁数の選択
０ １ ０ １	０５Ｈ	テキスト４０字モード	
０ １ １ ０	０６Ｈ	ＡＮＫは６×８ドット	ＡＮＫ文字の文字パターンの選択
０ １ １ １	０７Ｈ	ＡＮＫは７×１３ドット	
１ ０ ０ ０	０８Ｈ	高解像度モード	グラフィック画面を低解像度に見せるかどうかの選択
１ ０ ０ １	０９Ｈ	縦２００ラインモード	
１ ０ １ ０	０ＡＨ	コードアクセスモード	ＫＣＧアクセスモードの選択
１ ０ １ １	０ＢＨ	ビットマップモード	
１ １ ０ ０	０ＣＨ	不揮発メモリ書き込み不可	不揮発メモリの書き込みモードの選択
１ １ ０ １	０ＤＨ	不揮発メモリ書き込み可	
１ １ １ ０	０ＥＨ	画面表示不可	テキスト画面・グラフィック画面を含んだ全体の画面表示の有無の選択
１ １ １ １	０ＦＨ	画面表示可	

ボーダーカラーというのは，CRT画面のうち画面表示が可能な領域の周りの，画面表示が不可能な領域に付ける色を設定するものです．Iで明るさ（1のとき明るい色），Gで緑要素のありなし（1のときあり），Rで赤要素のありなし，Bで青要素のありなしを指定します．G，R，Bがどのようになったときボーダーカラーがどの色になるかは，図2-25の右側の表の通りです．ただし，MATEを除くノーマル98では，ボーダーカラーが指定できるのは標準解像度ディスプレイ（水平周波数15KHz）を使っていると

きのみで，しかも明るさの指定はできません．H98では，すべての解像度のディスプレイについてボーダーカラーを指定することができますが，明るさの指定はグラフィックモードが16色モードのときのみ有効となります．

たとえば，ボーダーカラーとして赤を指定したいときには，

```
outportb(0x6C, 0x20);
```

とします．

このほか，テキスト関係のI/Oポートには，I/Oアドレス70H～7AHのCRTC，A1H～A9HのKCGがありますが，これらについては関係各項を参照してください．

■2-6-3 テキストGDC

テキストGDCは，CRTCとともにテキスト画面やCRTディスプレイの制御などを行っているLSIです．ソフトウェアから見たときのテキストGDCの主な役割としては，

・CRTの水平・垂直同期信号の周期や幅の決定
・1画面の表示文字数・表示ライン数の決定
・VRAMの構成や表示開始番地の決定
・カーソルの表示とその形状の決定

等があります．

テキストGDCを制御するためのI/Oポートを表2-24に示します．テキストGDCを制御するためには，まずI/Oポート62Hにコマンドコードを出力します．それから，そのコマンドに付随するパラメータをI/Oポート60Hに順に出力していきます．このパラメータは，必ずしも全部与える必要はなく，途中までしか与えなかった場合には，それ以降の部分は前の値が引き継がれます．

表2-24　GDC関係のI/Oポート

リード／ライト	I／O アドレス	機　　能
リード	60H	GDCステータスの読み出し
	62H	GDCデータの読み出し
ライト	60H	GDCパラメータの書き込み
	62H	GDCコマンドの書き込み

このコマンドとパラメータの書き込みを行うときには，GDC側の受け入れ態勢について考慮してやることが必要です．GDCはコマンドとパラメータの受け取りが遅いため，CPUから送られてくるデータをいったんFIFOという一種のバッファに蓄えてから処理しています．そのため，多少連続してデータを出力しても問題は起こりませんが，このFIFOは16バイト分しかないので，あまり連続して出力するとFIFOがいっぱいになって，データが受け付けられなくなってしまいます．そこで，GDCにコマン

ドやパラメータをある程度連続して出力するような場合には，GDCのステータスレジスタからFIFOの状況を読み取り，FIFOが溢れてしまわないように出力してやる必要があります．GDCのステータスレジスタの形式は表2-25の通りです．

表2-25　GDCのステータスレジスタ（I/Oアドレス60H）

フラグ名	桁	意　　　味
DATA READY	D0	1のとき，データ読み出し系のコマンドを実行した後，データが読み出し可能になったことを示します．
FIFO FULL	D1	1のとき，ＦＩＦＯがいっぱいになっていて，コマンドやデータを受けつけられない状態であることを示します．
FIFO EMPTY	D2	1のとき，ＦＩＦＯが空になっていることを示します．
DRAWING	D3	1のとき，ＧＤＣが描画動作を行っている最中であり，ＣＰＵがＶＲＡＭにアクセスしてはならないことを示します．*
DMA EXECUTE	D4	1のとき，ＧＤＣがＤＭＡ転送を実行している最中であることを示します．**
VERTICAL SYNC	D5	1のとき，垂直同期信号（ＶＳＹＮＣ）が発生していて，垂直同期期間中であることを示します．
HORIZONTAL BLANK	D6	1のとき，水平消去信号が発生していることを示します．
LIGHT PEN DETECT	D7	1のとき，ライトペン信号によるアドレスが検出されたことを示します．**

*グラフィックGDCでのみ有効．　**98では意味を持たない．

FIFOが溢れないように出力する具体的な方法としては，出力する前に毎回FIFOに空きがあるかチェックする方法と，FIFOが空になったことを確認してから最大16バイト分をまとめて出力する方法が考えられますが，速度などの面で後者の方が有利なことが多いようです．なお，パラメータがないコマンドや少ししかないコマンドを，散発的に出力するだけの場合なら，これらのような配慮は無用です．

テキストGDCのコマンドと各コマンドのときに与えるパラメータを表2-26にまとめて示しておきます．以下では，表中の各コマンドの，テキストGDCで有効なものについての説明をしようと思います．なお，パラメータ表の中のH，M，Lの添字は，それぞれ上位，中位，下位のデータであることを示します．

表2-26　GDCのコマンド・パラメーター覧

■GDC動作制御コマンド

コマンド	動作内容	C/P	コード							
			DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
RESET	初期化動作	C	0	0	0	0	0	0	0	0
SYNC	動作モード，同期信号波長の定義	C	0	0	0	0	1	1	1	DE
		P1	0	0	CHR	F	I	D	G	S
		P2	← C/R →							
		P3	← VS_L →			← HS →				
		P4	← HFP →						← VS_H →	
		P5	0	0	← HBP →					
		P6	0	0	← VFP →					
		P7	← L/F_L →							
		P8	← VBP →						← L/F_H →	
MASTER／SLAVE	マスタ動作，スレーブ動作の選択	C	0	1	1	0	1	1	1	M

■GDC表示制御コマンド

コマンド	動作内容	C/P	コード							
			DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
START*	表示の開始の指示	C	0	1	1	0	1	0	1	1
			0	0	0	0	1	1	0	1
STOP	表示の停止の指示	C	0	0	0	0	1	1	0	0
ZOOM	拡大表示係数，拡大描画係数の設定	C	0	1	0	0	0	1	1	0
		P	← ZR →				← ZW →			
SCROLL	表示開始アドレス表示領域の設定	C	0	1	1	1	← RA →			
		P1 ⋮ P16	内蔵データRAMのフォーマットに従ってパラメータを設定							
CSRFORM	文字表示時のカーサ形状等の設定	C	0	1	0	0	1	0	1	1
		P1	CS	0	0	← L/R →				
		P2	← BL_L →		BD	← CST →				
		P3	← CFI →					← BL_H →		
PITCH	映像メモリ水平方向ワード数の設定	C	0	1	0	0	0	1	1	1
		P1	← P →							
LPEN*	ライトペン・アドレスの読み出し指示	C	1	1	0	0	0	0	0	0
			← LAD_L →							
			← LAD_M →							
			×	×	×	×	×	×	← LAD_H →	

*コマンドコードは6BHまたは0DHの8ビットを使用します

**LPENコマンドの発行後LAD_L，LAD_M，LAD_H，の順にCPUが読み出すことができます

■GDC描画制御コマンド

コマンド	動作内容		C/P	コード							
				DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
VECTW	描画に必要な各種パラメータの設定		C	0	1	0	0	1	1	0	0
			P1	SL	R	C	T	L	← DIR →		
			P2	← DC$_L$ →							
			P3	×	DGD	← DC$_H$ →					
			P4	← D$_L$ →							
			P5	×	×	← D$_H$ →					
			P6	← D2$_L$ →							
			P7	×	×	← D2$_H$ →					
			P8	← D1$_L$ →							
			P9	×	×	← D1$_H$ →					
			P10	← DM$_L$ →							
			P11	×	×	← DM$_H$ →					
VECTE	直線，四辺形，円弧，1ドット描画の実行の指示		C	0	1	1	0	1	1	0	0
TEXTW	グラフィック・テキスト・コード設定		C	0	1	1	1	1	← RA →		
			P1	← TX8，PTN$_L$ →							
			P2	← TX7，PTN$_H$ →							
			P3	← TX6 →							
			P4	← TX5 →							
			P5	← TX4 →							
			P6	← TX3 →							
			P7	← TX2 →							
			P8	← TX1 →							
TEXTE	グラフィック・テキスト描画実行指示		C	0	1	1	0	1	0	0	0
CSRW	描画アドレスの設定	文字モード	C	0	1	0	0	1	0	0	1
			P1	← EAD$_L$ →							
			P2	0	0	0	← EAD$_H$ →				
		文字／グラフィック混在モードで文字表示／描画	C	0	1	0	0	1	0	0	1
			P1	← EAD$_L$ →							
			P2	← EAD$_H$ →							
		文字／グラフィック混在モードで文字表示／描画	C	0	1	0	0	1	0	0	1
			P1	← EAD$_L$ →							
			P2	← EAD$_H$ →							
			P3	← dAD →				0	0	0	0
		グラフィック・モード	C	0	1	0	0	1	0	0	1
			P1	← EAD$_L$ →							
			P2	← EAD$_M$ →							
			P3	← dAD →				0	0	← EAD$_H$ →	

CSRR*	描画アドレスの読み出し指示	C	1	1	1	0	0	0	0	0
			← EAD$_L$ →							
			← EAD$_M$ →							
			×	×	×	×	×	×	← EAD$_H$ →	
			← dAD$_L$ →							
			← dAD$_H$ →							
MASK	マスク・レジスタ値の設定	C	0	1	0	0	1	0	1	0
		P1	← MASK$_L$ →							
		P2	← MASK$_H$ →							

*CSRRコマンドの発行後EAD$_L$，EAD$_M$，EAD$_H$，dAD$_L$，dAD$_H$，の順にCPUが読み出すことができます。

■GDC映像メモリ制御コマンド

			コード							
			DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
WRITE	パラメータの映像メモリへの書き込み準備	C	0	0	1	← WLH →		0	← MOD →	
		P1 ⋮								
READ	映像メモリ・データの読み出し指示	C	1	0	1	← WLH →		0	← MOD →	
DMAW	映像メモリへのDMA転送開始の指示	C	0	0	1	← WLH →		1	← MOD →	
DMAR	映像メモリからのDMA転送開始の指示	C	1	0	1	← WLH →		1	← MOD →	

1 RESET /GDC動作制御

コマンドコード

0	0	0	0	0	0	0	0

（00H）

動作

GDCを初期化します．

使用例

C言語でテキストGDCをリセットするには，

```
outportb(0x62,0x00);
```

とします．

2 SYNC /GDC動作制御

コマンドコード

0	0	0	0	1	1	1	DE

パラメータ

P1	0	0	CHR	F	I	D	G	S
P2	←C／R→							
P3	←VSL→			←HS→				
P4	←HFP→						←VSH→	
P5	0	0	←HBP→					
P6	0	0	←VFP→					
P7	←L／FL→							
P8	←VBP→						←L/FH→	

動作

GDCの全体的なモード設定（動作モード，同期信号の周期・幅等の設定）を行います．表中のパラメータの意味は以下の通りです．なお，このコマンドは画面表示をめちゃめちゃにしてしまう可能性がある危険なコマンドですので，使い方を十分に理解してから使うようにしてください．

◆DE：表示の許可・禁止の指定

DE	意　味
0	表示を禁止する
1	表示を許可する

以下のパラメータを設定するときには，画面の乱れを防ぐため，通常は表示禁止状態にしておきます．

◆CHR，G：文字モード・グラフィックモードの選択

CHR	G	意　味
0	0	文字・グラフィック混在モード
0	1	グラフィックモード
1	0	文字モード
1	1	設定不可

テキストGDCは文字・グラフィック混在モード，グラフィックGDCはグラフィックモードとします．

◆F：描画タイミングの指定

F	意　味
0	フラッシュ描画
1	フラッシュレス描画

表示期間中も描画を行うかどうかを指定します．フラッシュ描画のとき，表示期間中も描画を行います．グラフィックGDCのとき，VRAMがデュアルポートRAMの機種ではフラッシュ描画とします．そうでない機種でフラッシュ描画をすると画面がちらつきます．テキストGDCでは意味を持ちません．

◆I，S：インタレース走査の指定

I	S	意　味
0	0	ノンインタレース
0	1	設定不可
1	0	インタレース
1	1	インタレース・シュリンク

I，Sはテキスト・グラフィックGDCに同じ値を設定します．ノーマルモードでは通常ノンインタレースを設定します．

◆D：メモリリフレッシュ動作の有無の指定

F	意　味
0	リフレッシュ動作なし
1	リフレッシュ動作あり

GDCがメモリリフレッシュを行うかどうかを指定します．テキストGDCはリフレッシュ動作なし，グラフィックGDCはリフレッシュ動作ありとします．

◆C/R：1行当たりの表示文字数の指定

1行当たりの文字数−2を指定します．文字数は，偶数のみ指定可能です．テキストGDCで80桁の場合は，80−2=78（4EH）を，グラフィックGDCではノーマルモードのときは16ドットを1文字として，40−2=38（26H）を指定します（2.5MHzの場合）．

◆HS：水平同期信号の幅の指定

水平同期信号の幅を文字数に換算したときの，文字数−1を指定します．最低2文字分（設定値では1以上）は必要です．

◆HFP：CRT管面右方の非表示区間の指定

画面表示の右の表示がされない部分の幅を文字数換算で指定します．文字数−1で指定しますが，4文字以上（設定値では3以上）必要です．

◆HBP：CRT管面左方の非表示区間の指定

画面表示の左の表示がされない部分の幅を文字数換算で指定します．文字数−1で指定しますが，3文字以上（設定値では2以上）必要です．

◆VS：垂直同期信号の幅

垂直同期信号の幅を，ライン数換算で指定します．指定した数が，ライン数そのものになります．0は指定できません．

◆VFP：CRT管面下方の非表示区間の指定

画面表示の下の表示がされない部分の幅をライン数換算で指定します．指定した数がライン数そのものになり，0は指定できません．

◆VBP：CRT管面上方の非表示区間の指定

画面表示の上の表示がされない部分の幅をライン数換算で指定します．指定した数がライン数そのものになり，0は指定できません．

◆L/F：1画面当たりの表示ライン数

1画面に表示するライン数を指定します．指定した値がライン数そのものになり，0を指定すると1024ラインとなります．ノーマルモードでは通常400ラインを指定します．

なお，同期信号関係のパラメータを指定するときには，テキストGDCとグラフィックGDCの設定値が矛盾しないようにしなければなりません．すなわち，水平同期関係の値は，グラフィックGDCの文字数換算の値が，GDC 2.5MHzモードのときはテキストGDCのそれの1／2になるように，GDC 5MHzモードのときはテキストGDCのそれと等しくなるように，さらに，垂直同期関係の値はグラフィックGDCのライン数換算の値がテキストGDCのそれと常に一致するように設定しなければなりません．なお，各パラメータのノーマルモード・高解像度ディスプレイでの標準的な設定値を表2-27に示しておきます．

表2-27　SYNC命令パラメータの標準的設定値

信号名	設定値		
	テキスト	グラフィック（2.5MHz）	グラフィック（5MHz）
C／R	4EH	26H	4EH
HS	07H	03H	07H
HFP	09H	04H	09H
HBP	07H	03H	07H
VS	08H	08H	08H
VFP	07H	07H	07H
VBP	19H	19H	19H
L／F	190H	190H	190H

使用例　サンプルプログラム（p.99）参照

■表示制御系コマンド

1 START　/テキストGDC表示制御

コマンドコード

0	1	1	0	1	0	1	1

（6BH）

あるいは

0	0	0	0	1	1	0	1

（0DH）

動作　画面表示の開始を指示します．

使用例　C言語でテキスト画面の表示を開始するには，

```
outportb(0x62,0x0d);
```

とします．

2 STOP　/テキストGDC表示制御

コマンドコード

0	0	0	0	1	1	0	0

（0CH）

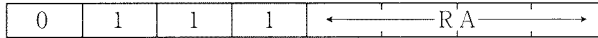
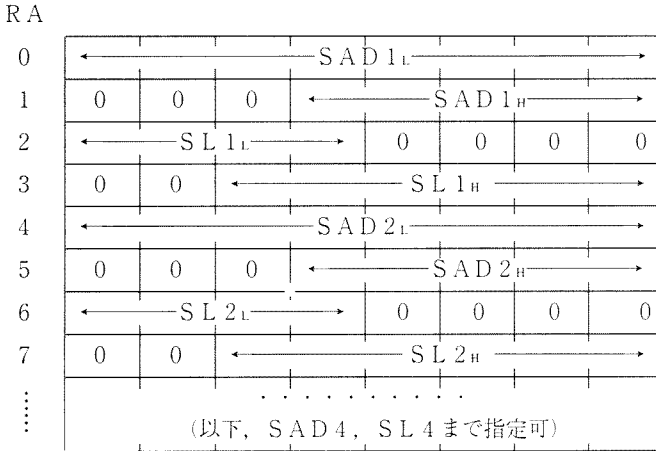
動作　画面表示の停止を指示します．

使用例　C言語でテキスト画面の表示を停止するには，

```
outportb(0x62,0x0c);
```

とします．

3 SCROLL/GDC表示制御

コマンドコード　| 0 | 1 | 1 | 1 | ←RA→ |

パラメータ　RA

動作　表示画面の分割，各表示画面の表示開始アドレスの指定や表示ライン数の設定を行ないます．各パラメータの意味は次の通りです．

◆RA：書きかえ開始パラメータ位置の指定

このコマンドには多くのパラメータ（最大16個）があり，その都度全部のパラメータを書きかえるのではたいへんなので，このRAの部分に書きかえ始めるパラメータの位置を指定します．書きかえなかったパラメータには前の値がそのまま残ります．第1パラメータから書きかえ始めるときには，0を指定します．

◆SAD：表示開始アドレスの指定

各画面の表示開始アドレスを指定します．ここに指定するアドレスは，CPUから見たアドレスではなく，GDCから見たアドレスですので注意してください．GDCから見たアドレスとは，VRAMの先頭番地を0として1アドレス1ワード（16ビット）のアドレス，つまりCPUのアドレスに比べ進み方が1/2のアドレスです．

◆SL：表示ライン数の指定

各画面の表示ライン数を指定します．各画面の表示ライン数の合計がL/F（SYNCコマンド参照）の値以上になるようにします．通常のように1つの画面を普通に表示するなら，L/Fと同じ値を指定します．

使用例　C言語でテキスト画面の表示開始位置をVRAM上の5行目からにするには，GDCのアドレスでは1行が50Hであり，表示開始アドレスは50H×5=190Hとなるので，

```
outportb(0x62,0x70);
outportb(0x60,0x90);
```

```
outportb(0x60,0x01);
```

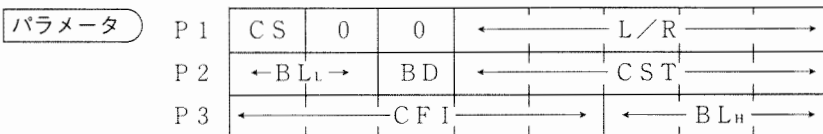
とします．この表示開始アドレスを800Hにすれば，裏VRAMの表示をすることができます．

4 CSRFORM /GDC表示制御

コマンドコード

0	1	0	0	1	0	1	1	(4BH)

パラメータ

P1	CS	0	0	←L/R→				
P2	←BL_L→		BD	←CST→				
P3	←CFI→					←BL_H→		

動作

1行のライン数の設定や，カーソルの制御を行います．各パラメータの意味は以下の通りです．

◆L/R：1行中のライン数の指定

1行に含まれるライン数−1を指定します．ノーマルモードでは通常，25行モードのとき15（16ライン）が設定されています．

◆CS：カーソル表示の有無の指定

CS	意　　味
0	カーソルを表示しない
1	カーソルを表示する

◆BD：カーソル点滅の有無の指定

BD	意　　味
0	カーソルを点滅させる
1	カーソルを点滅させない

◆BL：カーソル点滅周期の指定

カーソルの点滅周期を設定します．指定値が小さな値であるほどカーソルの点滅は速くなりますが，0は指定できません．通常は0CHが設定されています．

◆CST：カーソル表示開始ラインの指定

カーソルの行の中での上の端の位置を，行の一番上のラインから数えて何ライン目かの数値−1で指定します．通常は0（1ライン目）が設定されています．

◆CFI：カーソル表示終了ラインの指定

カーソルの行の中での下の端の位置を，行の一番上のラインから数えて何ライン目かの数値−1で指定します．CFI≦L/Rでなければなりません．このパラメータと上のCSTによって，カーソルの縦方向の大きさが決まります．通常はCFI=L/Rとなる値（25行モ

ードのとき0FH）が設定されています．

使用例　C言語でカーソルを高速点滅のアンダーラインカーソルにするには，L/R=0FH，CS=1，BD=0，BL=02H，CST=0FH，CFI=0FHと設定すればよいので，

```
outportb(0x62,0x4b);
outportb(0x60,0x8f);
outportb(0x60,0x8f);
outportb(0x60,0x78);
```

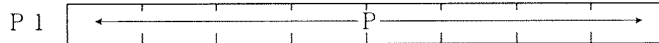
とします（25行モードの時）．

5 PITCH／GDC表示制御

コマンドコード

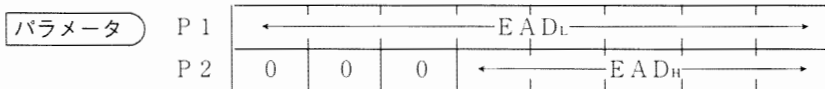
0	1	0	0	0	1	1	1

（47H）

パラメータ　P1　←――P――→

動作　VRAMの横幅を，文字数で指定します．指定した値がそのまま文字数になります．これをC/R（SYNCコマンド参照）よりも小さく取ると同じデータが繰り返し使われ，大きく取るとVRAMの一部だけを表示することができます．ノーマルモードでは通常，80（50H）が設定されています．

使用例　C言語でテキストVRAMの行を1行おきに飛ばして表示するようにするには，Pの値として通常の50Hの2倍のa0Hを指定すればよいので，

```
outportb(0x62,0x47);
outportb(0x60,0xa0);
```

とします．

動作　カーソル位置の設定を行います．パラメータの意味は次の通りです．

◆EAD：カーソル位置の指定

カーソルの位置をGDCアドレスで指定します．

使用例　C言語で（x,y）の位置にカーソルを設定するためには，adrを整数変数として，

```
adr = y * 80 + x;
 outportb(0x62,0x49);
 outportb(0x60,adr % 0x100);
 outportb(0x60,adr / 0x100);
```

とします（表示開始番地が変更されていないとき）．

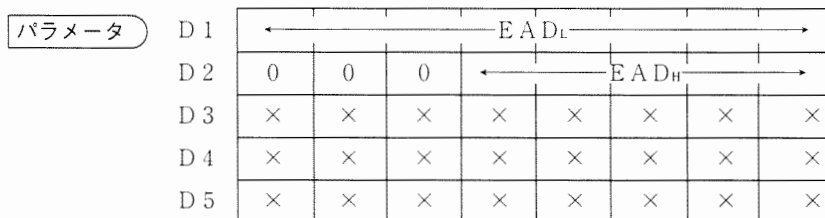
2 CSRR /GDC描画制御

コマンドコード

1	1	1	0	0	0	0	0

(E0H)

パラメータ

D1	←EADL→							
D2	0	0	0	←EADH→				
D3	×	×	×	×	×	×	×	×
D4	×	×	×	×	×	×	×	×
D5	×	×	×	×	×	×	×	×

動作　カーソルの現在位置を読み出します．このコマンドを発行した後，データレジスタ（I/Oアドレス62H）からEADL，EADHの順に読み出すことができます．EADの意味はCSRWコマンドと同じです．テキストGDCでは，D3～D5は意味のない数値を返してきますので，これらは読み捨てるようにします．

なお，このコマンドをいったん発行したら，たとえ読み出す必要がなくなったとしても必ず5バイトのデータを読み切るようにしてください．さもないと，いつまでもGDC側からのデータがFIFOに残ったままになってしまうので，その後でFIFOが空になるのを待つようなプログラムを実行するとハングアップしてしまう可能性があります．

使用例　C言語でカーソルの現在位置を取得するには，

```
int x,y,i,j,adr;
outportb(0x62,0xe0);          /*CSRRコマンド*/
for (i = 1; i <= 3; i++);   /*リカバリタイム*/
while ((inportb(0x60) & 1) == 0); /*コマンド終了の検出*/
adr  = inportb(0x62);       /*EADL*/
adr += inportb(0x62) * 0x100;      /*EADH*/
for (j = 1; j <= 3; j++)    /*読み捨て*/
     inportb(0x62);
x = adr % 80;
y = adr / 80;
```

としてやれば，xにカーソルのX座標が，yにカーソルのY座標が得られます（表示開始番地が変更されていないとき）．

■サンプルプログラム

このサンプルプログラムは，テキスト画面の表示を，通常25行表示が最大であるものを30行表示にするプログラムです．その原理は，次のようなものです．テキストVRAMの容量は表裏画面合わせて51行分あり，これらが連続したアドレスに存在しているので，VRAMの容量だけからいえば理論上最大51行までの表示が可能です．そこで，問題は具体的にどうやって多行表示を実現するかということになりますが，これには次の2つの方法が考えられます．

1）1行に含まれるライン数（L/R）を減らす
2）1画面に表示するライン数（L/F）を増やす

このうち，1）の方法はCSRFORMコマンドでテキストGDCのL/Rを減らし，CRTCのボディフェイスライン数もそれに合わせて変えてやることで実現できます．ただし，それだけだと表示文字の下の方が切れてしまいますので，たとえば1行12ラインにしたなら，縦12ドットのフォントをユーザー定義文字で作ってやるなどの工夫も必要になります．

このプログラムでは2）の方法を使っています．つまり，L/Rは16ラインのまま30行表示を実現するために，SYNCコマンドを使ってL/Fを16×30=480にしているのです．ここで注意しなければならないのは，テキストとグラフィックのGDCはいつも同期していなければならないので，両方のGDCについてL/Fを変更してやる必要があることです．

この方法を使えば，解像度そのものが増すので，すっきりと30行表示が実現できるように思えるのですが，実はこの方法にも欠点はあります．この方法では，水平同期周波数はそのままでL/Fを増やすことになるので，垂直同期周波数が若干下がってしまうのです．その結果，1秒当たりのスキャン回数が少なくなるので，画面が若干ちらつきます．それなら，水平同期周波数を上げればいいじゃないかということになりますが，残念ながら普通の98では水平同期周波数を上げることはほとんど不可能です．それが可能なのはMULTIやMATEのみです．したがって，普通の98ではちらつきの発生は避けられません．このちらつきがどれくらい気になるかは，個人差があると思います．実際に見てみてください．

こうしてせっかく30行表示をしても，DOSは25行までしか関知しませんから，そのままでは26行目以

降が表示されているかどうかわかりません．そこで，このプログラムでは，余分に表示されるようになった部分のVRAMに文字コードを直接書き込んで，表示されていることが確認できるようにしてあります．

なお，このプログラムはGDC 2.5MHzモード用です．GDC 5MHzモードの場合には，グラフィックGDCの水平同期関係のパラメータを変更する必要があります．

```
#include <stdio.h>
#include <dos.h>

void main(void)
{
        int i, j;

        while((inportb(0x60) & 4) == 0);     /*ＦＩＦＯチェック*/
        outportb(0x62, 0x0e);               /*テキストＧＤＣ　ＳＹＮＣコマンド*/
        outportb(0x60, 0x00);
        outportb(0x60, 0x4e);
        outportb(0x60, 0x07);
        outportb(0x60, 0x25);
        outportb(0x60, 0x07);
        outportb(0x60, 0x07);
        outportb(0x60, 0xe0);               /*Ｌ／Ｆ＝４８０（０１Ｅ０Ｈ）*/
        outportb(0x60, 0x65);
        outportb(0x62, 0x47);               /*テキストＧＤＣ　ＰＩＴＣＨコマンド*/
        outportb(0x60, 0x50);
        outportb(0x62, 0x70);               /*テキストＧＤＣ　ＳＣＲＯＬＬコマンド*/
        outportb(0x60, 0x00);
        outportb(0x60, 0x00);
        outportb(0x60, 0x00);               /*ＳＬ１＝４８０（０１Ｅ０Ｈ）*/
        outportb(0x60, 0x1e);

        while((inportb(0xa0) & 4) == 0);     /*ＦＩＦＯチェック*/
        outportb(0xa2, 0x0e);               /*グラフィックＧＤＣ　ＳＹＮＣコマンド*/
        outportb(0xa0, 0x06);
        outportb(0xa0, 0x26);
        outportb(0xa0, 0x03);
        outportb(0xa0, 0x11);
        outportb(0xa0, 0x03);
        outportb(0xa0, 0x07);
        outportb(0xa0, 0xe0);               /*Ｌ／Ｆ＝４８０（０１Ｅ０Ｈ）*/
        outportb(0xa0, 0x65);
        outportb(0xa2, 0x47);               /*グラフィックＧＤＣ　ＰＩＴＣＨコマンド*/
        outportb(0xa0, 0x28);
        outportb(0x62, 0x0d);               /*テキストＧＤＣ　ＳＴＡＲＴコマンド*/
        outportb(0xa2, 0x0d);               /*グラフィックＧＤＣ　ＳＴＡＲＴコマンド*/
        for (i = 25; i < 30; i++)           /*画面を拡張した部分への書き込み*/
           for (j = 0; j < 80; j++) {
              poke(0xa000, i * 160 + j * 2, '1' + i - 25); /*数字*/
              poke(0xa200, i * 160 + j * 2, 0x20 * (i - 22) + 1);  /*色*/
           }
}
```

CRTCは，テキストGDCとともにテキスト画面表示の制御を行っているLSIです．CRTCのソフトウェアから見た主な役割は，CG（キャラクタジェネレータ）から読み出したパターンのどの部分を表示するかを決定することと，テキスト画面のスムーススクロール（1ドット単位でのスクロール）を実現することです．

CRTCを制御するI/Oポートの一覧を表2-28に示します．このうち，上の3つはCGから読み出したパターンのどの部分を表示するかの指定，下の3つはスムーススクロールの指定とどの部分をスクロールさせるかの指定に用います．以下に，これらの値についての詳しい説明を行います．なお，以下の説明はノーマルモードの場合のもので，通常セットしておく値として示してあるのは，特に断らないかぎり25行モードのときの値です．20行のときの値は若干異なります．

表2-28　CRTCを制御するI/Oポート

リード／ライト	I／O アドレス	機　　能
ライト	70H	キャラクタ位置ライン数の書き込み
	72H	ボディフェイスライン数の書き込み
	74H	キャラクタライン数の書き込み
	76H	スムーススクロールライン数の書き込み
	78H	スクロールエリア上辺位置行数の書き込み
	7AH	スクロールエリア行数の書き込み

1）キャラクタ位置ライン数（I/Oアドレス70H）

キャラクタ位置ライン数というのは，CRTCがCGから読み出した文字パターン（通常16ライン分）のどの部分から表示を始めるかを，表示を始めるライン数−1で指定するものです．たとえば，ここに8を指定すると，文字のほぼ真ん中（9ライン目）から表示が始まるようになるので，文字の上半分は表示されなくなります．通常は，ここには0をセットしておきます．

2）ボディフェイスライン数（I/Oアドレス72H）

ボディフェイスライン数というのは，CRTCがCGから読み出した文字パターンをどこまで表示するかを，最後に表示するライン数−1で指定するものです．たとえば，ここに12を指定すると，文字パターンの13ライン目より下は表示されなくなります．通常は，ここには15（0FH）をセットしておきます．

このキャラクタ位置ライン数とボディフェイスライン数によって，1行の中に占めるキャラクタのライン数は，

（ボディフェイスライン数）−（キャラクタ位置ライン数）＋1

となります．この値は通常L/R（1行に含まれるライン数）と同じにしますが，これがL/Rよりも小さな

値だった場合には，1行の中に同じ文字パターンのラインが繰り返し現れ，ラインのカウント数がはんぱになった分は次の行にも影響を与えます．また，キャラクタ位置ライン数，ボディフェイスライン数はともに下位5ビットのみが有効であり（最大1FH），1FH（32ライン目）の次は00H（1ライン目）となります．したがって，1EH（31ライン目）から0DH（14ライン目）までの16ライン，といった指定も可能です．また，2つの値によって指定された領域が文字パターンの16ライン目より下の部分を含んだ場合，その部分の表示は空白になります．

3）キャラクタライン数（I/Oアドレス74H）

キャラクタライン数というのは，CRTCがCGから読み出した文字パターンをどこまで有効にするかを，有効な最後のラインのライン数−1で指定するものです．たとえば，ここに8を指定すると，9ライン目より下のパターンがマスクされ，文字の下半分は表示されず，空白になります．通常は，ここには15（0FH）をセットしておきます．

4）スムーススクロールライン数（I/Oアドレス76H）

スムーススクロールライン数というのは，スクロールエリアの中の文字を上にずらすライン数を指定するものです．たとえば，ここに5を指定すると，スクロールエリア内の文字がすべて5ライン分だけ上にずれて表示されます．ずらせる幅は，最大１５ライン分までですが，このスムーススクロールライン数で15ライン分までずらしておいて，ソフトウェア的に，あるいはGDCによって1行分（16ライン分）上にずらしてからスムーススクロールライン数を0に戻す，という操作を繰り返すことによって，任意のドット数のスムーススクロールを実現することも可能です．

5）スクロールエリア上辺位置行数（I/Oアドレス78H）

スクロールエリア上辺位置行数というのは，テキスト画面のうちスムーススクロールさせたい部分の上端の位置を，一番上の行を0とした行数によって指定するものです．画面全体をスクロールさせたいときには0を指定します．

6）スクロールエリア行数（I/Oアドレス7AH）

スクロールエリア行数の部分には，スクロールさせたい領域の行数−1を指定します．画面全体をスクロールさせたいときには，25行モードのときは24（18H）を，20行モードのときは 19(13H) を指定します．

■サンプルプログラム（ノーマルモードのみ）

このサンプルプログラムは，CRTCとGDCのスクロール機能を使って，テキスト画面のスムーススクロール，つまり1ライン（1ドット）単位のスクロールを行うプログラムです．原理的には本当に1ラインずつスクロールさせることも可能ですが，それではスクロール速度が遅すぎるので，実際には4ドットずつスクロールさせています．そのようにして滑らかに表示領域を裏画面に移し，それから再び表画面に滑らかに表示を移します．このプログラムに含まれている関数textscは，引数として表示を開始するライン数を指定すると，CRTC，GDCをコントロールして，そのライン数から表示を始めるように

する関数です．

具体的なスクロールの方法としては，まず，スクロールをする前に垂直同期信号が発生するのを待ちます．これは，CRTの1スキャン中に2回以上スクロールさせてしまうのを防ぐ，等の目的のためです．それから，GDCのSCROLLコマンドによって16ライン（1行）単位の表示位置の設定を，CRTCのスムーススクロールライン数によって1ライン単位の表示位置の設定をします．

このプログラムでは，スクロールによって裏画面が出てくるようにしましたが，これをラップラウンドスクロール（ある行数までいくと再び1行目に戻ってくるスクロール）にしようとすると，GDCのSCROLLコマンドで2画面分割をするなどの工夫が必要になってきます．その具体的な方法についてはみなさんが考えてみてください．

```
#include <stdio.h>
#include <dos.h>

void textsc(int);

void main(void)
{
   int i;

   outportb(0x78, 0);  /*スクロールエリア上辺位置＝０*/
   outportb(0x7a, 24); /*スクロールエリア行数＝２５*/

   for (i = 0; i < 80 * 25; i++) {    /*裏ＶＲＡＭへの書き込み*/
   poke(0xa000, 160 * 25 + i * 2, i % 0x100);
   poke(0xa200, 160 * 25 + i * 2, 0xe1);
   }

   for (i = 0; i <= 4 * 25; i++) textsc(i * 4); /*アップスクロール*/
   for (i = 4 * 25; i >= 0; i--) textsc(i * 4); /*ダウンスクロール*/
}

void textsc(int lin)    /*ライン単位での表示位置設定関数*/
{
   int gdcadr, smlin;

   gdcadr = lin / 16 * 80; /*ＧＤＣアドレス*/
   smlin = lin % 16;  /*スムーススクロールライン数*/
   while((inportb(0x60) & 0x20) != 0);  /*ＶＳＹＮＣ待ち*/
   while((inportb(0x60) & 0x20) == 0);
   while((inportb(0x60) & 4) == 0);     /*ＦＩＦＯチェック*/
   outportb(0x62, 0x70);      /*テキストＧＤＣ　ＳＣＲＯＬＬコマンド*/
   outportb(0x60, gdcadr % 0x100); /*ＳＡＤ１Ｌ*/
   outportb(0x60, gdcadr / 0x100); /*ＳＡＤ１Ｈ*/
   outportb(0x76, smlin);     /*スムーススクロールライン数セット*/
}
```

■2-6-5 KCGアクセス・ユーザー定義文字

●KCGアクセス

テキスト画面には，テキストVRAMに文字コード（キャラクタコード）を書き込むだけで文字を表示することができます．テキスト画面の表示を担当しているCRTCは，そのキャラクタコードを読み出して，それを文字パターンに変換してからそのパターンをテキスト画面に表示するわけですが，そのためには，CRTCにそれぞれの文字コードの文字の文字パターンを供給してやるハードが必要になります．その役割を担っているのがCG（キャラクタジェネレータ）で，このCGのうちで漢字の文字コードをCRTCに供給しているものを特にKCG（漢字キャラクタジェネレータ）と呼びます．ふだん，ごく普通に漢字をテキスト画面に表示するような場合には，ユーザーがこのKCGの存在を意識する必要はありませんが，漢字の文字パターンをCPUによって読み出してグラフィック画面に表示したいとか，自分で作った文字パターン（ユーザー定義文字）を登録してテキスト画面に表示したい，などといった場合には，このKCGを制御してやる必要があります．

KCGをコントロールするためのI/Oポートを表2-29に示します．これらのI/Oポートは基本的に，KCGとCPUとの間でデータをやり取りするためのものです．

表2-29　KCG関係のI/Oポート

リード／ライト	I／O アドレス	機能	D7	D6	D5	D4	D3	D2	D1	D0
リード	A9H	文字パターンの読み出し	←文字パターン→							
ライト	A1H	文字コード第2バイト書き込み	←文字コード第2バイト→							
	A3H	文字コード第1バイト書き込み	←文字コード第1バイト－20H→							
	A5H	文字パターン読み出し位置の指定（図2-26参照）	0	0	L／$\overline{R}$	RC4	RC3	RC2	RC1	RC0
	A9H	文字パターンの書き込み	←文字パターン→							

実は，これらのI/Oポートは普通，そのままの状態では使えません．なぜなら，このI/Oポートを使ってCPUがKCGにアクセスするということは，KCGという1つしかない資源に対して，CPUとCRTCが使用権を争うことになるからです（CRTCは文字表示のためにKCGをアクセスしている）．ふだん，KCGのアクセスモードはコードアクセスモードになっていますが，コードアクセスモードでは，画面表示を行っている間はCRTCがKCGの使用権を持っていて，その間はCPUがKCGをアクセスすることはできません．そのため，これらのI/Oポートを使ってKCGに対するデータの読み書きを行うためには，次の2通りの方法のうちのいずれかを取る必要があります．

1）モードフリップフロップ1（2-6-2．テキストのI/Oポート参照）を操作して，KCGアクセスモードをビットマップにしてやる．
2）KCGアクセスモードはコードアクセスのままで，VSYNC期間中にのみ読み書きを行う．

1）の方法は，表示期間中でも常にKCGの使用権をCPUに与えるものです．この方法を取れば，表示期間かどうかなどをいっさい気にしないでKCGに対するデータの読み書きができますが，CRTCがKCGに表示のためのアクセスをすることができなくなってしまうので，画面上に表示されていた漢字はすべてゴミに化けてしまいます．

2）の方法は，コードアクセスモードのままでも，画面表示をしていない期間（VSYNC信号が発生している期間）にはKCGの使用権が自動的にCPUに与えられることを利用しています．現在VSYNC期間中かどうかは，GDCのステータスレジスタのビット5を参照することで調べることができます．この方法を使えば，画面を乱すことなくKCGに対するデータの読み書きをすることができますが，VSYNCが発生している期間にしか読み書きができないので，読み書きに時間がかかるという欠点があります．結局，どちらの方法にも一長一短あるので，場合によって使い分けるのがベストでしょう．

なお，ANK文字の文字パターンについては，表示期間中は常にCRTCがアクセス権を持っているので，CPUがANK文字パターンを読み出せるのはKCGアクセスモードに関わらずVSYNC期間中のみです．

これらのような配慮をした上で，実際にKCGからデータを読み書きするためには，まず，KCGに読み書きする文字の文字コードを指定してやる必要があります．その文字コードはI/OアドレスA1HとA3Hに設定します．I/OアドレスA1Hには漢字のJISコードの下位バイトを，A3HにはJISコードの上位バイト−20Hを書き込みます．たとえば，JISコード3021Hの漢字（'亜'の字）なら，I/OアドレスA1Hには21Hを，A3Hには30H−20H=10Hを書き込んでやります．

次に，合計(16/8)×16=32バイトある漢字の文字パターンを，I/OアドレスA9Hに1バイトしかないI/Oポートから読み書きするために，文字パターンのどの部分を読み書きするかをI/OアドレスA5Hに指定してやります．ここに書き込んだ値と読み書きされるパターンの位置の関係については，図2-26を参照してください．ここに指定する値を順に変化させていきながらI/OアドレスA9Hを読み書きすることに

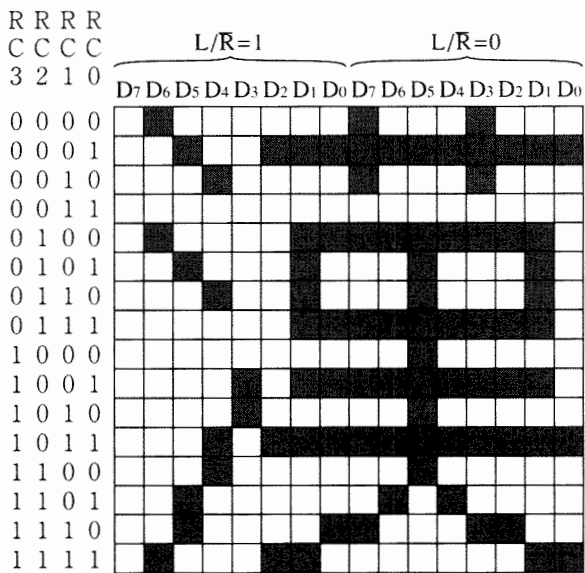



図2-26　I/OアドレスA5Hの各ビットの値とパターンの場所の関係

よって，文字パターンの全体を読み書きすることができます．

そして，パターンの読み書きが終わったなら，KCGアクセスモードをビットマップにしていた場合は，それをコードアクセスに戻して，CRTCにKCGのアクセス権を渡します．そうしないと，画面上の漢字はずっと化けたままになってしまいます．

このやり方が，初代98からのすべての98で行うことができるKCGアクセスの方法です．しかし，このやり方はかなり面倒ですし，時間もかかってしまいます．そこで，PC-9801VX辺りの機種からは，メモリ空間にCGウィンドウという領域が設けられ，I/OアドレスA1HとA3Hに文字コードをセットしさえすれば，あとはそのCGウィンドウから文字パターン全体を読み込めるようになりました（一部の文字では半分ずつの読み書きになる）．

CGウィンドウは，ノーマルモードでは絶対メモリアドレスA4000Hから存在しており，一般のメモリと同じようにアクセスすることができます．その形式は図2-27のようになっています．が，この図に示されているのはJIS第1，第2水準漢字の場合のもので，そのほかの文字（ユーザー定義文字など）の場合は，このうち奇数番地の部分のみが有効で，1度には文字の右半分か左半分しか読み書きできません（図2-28参照）．その場合，文字の左右どちらを読み書きするかは，I/OアドレスA5HのL/Rビットで指定します．L/R=0のとき右半分，L/R=1のとき左半分の読み書きとなります．

なお，このCGウィンドウからのアクセスでも，CPUがCRTCとKCGのアクセス権を争うことになってしまうのには変わりないので，I/Oのみでのアクセスのときと同じように，上に述べておいた2つのアクセス方法のいずれかを取る必要があります．それから，CGウィンドウのアクセススピードですが，機種によってばらつきはありますが，どの機種でもメインRAMより遅いことは確かです（メインRAMに比べてアクセスタイムは2～7倍程度かかる）．したがって，CGウィンドウにアクセスするときはできるだけ偶数番地からの16ビットアクセスをするべきです．ただ，CGウィンドウはたとえ32ビットマシンでも16ビットバスを通してつながっているので，32ビットアクセスをしてもあまり速くはなりません．

以上は，ノーマルモードでのKCGアクセスの方法です．ハイレゾモードでは，全機種にCGウィンドウが搭載されているので，常にCGウィンドウを通してのアクセスとなります．ハイレゾモードでは，CGウィンドウは絶対メモリアドレスE4000Hから存在しており，その形式は図2-29のようになっています．ここから文字パターンの読み書きをするには，ノーマルモードのときと同じように，I/OアドレスA1HとA3Hに文字コードを書き込み，CRTCとアクセスが衝突しないように，ノーマルモードのとき示した2つのアクセス方法のいずれかを取りながら読み書きを行うようにします．

●ユーザー定義文字

次に，ユーザー定義文字のことについて少し述べます．

ユーザー定義文字とは，ユーザーが文字パターンを自由に変更することができる文字のことです．文字の種類からすると漢字の部類に入る，2バイトの文字コードを持ち，普通画面上では2桁のスペースを占有する16×16ドットサイズの文字です．初代98にはユーザー定義文字はありませんでした．PC-9801E/F/MではJISコードの7621H～765FHの63文字が，それ以降の98では7621H～767EHと7721H～777EHの188文字が，ユーザー定義文字として使えるようになっています．ユーザー定義文字の文字パターンの登録のしかたは，KCGアクセスの説明の部分で書いた通りですが，そのほかにBIOSを使う方法やMS-DOSのUSKCGMコマンドを使う方法などがありますので，これらの方法を場合によって使い分

図2-27　CGウィンドウの形式（JIS第1，第2水準漢字の場合）

メモリアドレス　偶数番地　奇数番地
D7　D0 D15　D8
0A40h　00 02 04 06 08 0A 0C 0E 10 12 14 16 18 1A 1C 1E
不　定

図2-28　CGウィンドウの形式（JIS第1，第2水準以外の漢字の場合）

けるのがいいと思います．

上に述べたように，ユーザー定義文字の大きさは原則的には2桁（16×16ドット）ですが，1桁（8×16ドット）の大きさのユーザー定義文字がほしくなる場合もあると思います．その場合には，ユーザー定義文字を半角漢字と同じ形式でVRAMに書き込めば（2-6-1．テキストVRAM参照），ユーザー定義文字の左半分だけが表示され，半角漢字と同じように扱うことができます．ただし，ユーザー定義文字の右半分だけを表示することはできません．それに，半角で表示しようとする同じ文字コードのユーザー定義文字がたまたま2つ横に並ぶと，普通の大きさのユーザー定義文字として扱われてしまうので，ユーザー定義文字を半角表示するときは，文字パターンの左半分と右半分に同じパターンを書き込んでおくようにします．

なお，ユーザー定義文字はPC-9801E/F/Mを除いて188文字だと書きましたが，これはユーザー定義文字をJISコードの範囲に収めるための「たてまえ」で，実際には大部分の機種で256文字（このうちの数文字は表示できない）のユーザー定義文字が存在していて，VRAMに直接文字コードを書き込めばこれらを表示することができます．ただし，188文字以上の部分はメーカーが保証しているわけではない

図2-29 CGウィンドウの形式（ハイレゾモード）

ので，この部分を使うのはあまりおすすめできません．実際，この余分な部分の一部が使えない機種もあるようなので，よほどユーザー定義文字が足りなくなった場合以外は使わない方がよいと思います．

■サンプルプログラム（ノーマルモードのみ）

このサンプルプログラムは，KCGからJISコード3020H付近からの漢字の文字パターンを読み出し，それをグラフィック画面に表示するプログラムです．グラフィック画面のモードやパレットの設定には，C言語間のグラフィック命令の文法の違いへの配慮や，読者の方にグラフィックモードやパレットへの理解を深めていただきたいということから，あえてすべてをI/O直接制御にしてあります．グラフィックを扱うときの参考にしてください．

このプログラムに含まれている関数ですが，関数inpkanは指定された文字コードの文字の文字パターンを読み出す関数，関数setpalは16色モードでのカラーパレットを設定する関数です．これらを用いて，このプログラムでは，パレット番号6の色を茶色に設定しておいて，読み出した文字パターンをパレット番号６でグラフィックVRAMに書き込むことで，茶色い漢字を表示しています．

このプログラムは，CGウィンドウを使っていないので，CGウィンドウがない機種でも動作します．また，基本的には16色モード搭載機種で動作することを想定していますが，8色モードでも動作するように細工してあるので，16色モードが搭載されていない機種でもほぼ正常に動作します．

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>

void inpkan(int, char *);
void setpal(int, int, int, int);

void main(void)
{
  int i, j, k, kcod, vadd, kpat;
  char kanpt[32];

  clrscr();
  outportb(0x68, 0x08);        /*４００ラインモード指定*/
  outportb(0xa2, 0x4b);        /*ＧＤＣ　ＣＳＲＦＯＲＭコマンド*/
  outportb(0xa0, 0);           /*Ｌ／Ｒ＝１（４００ラインモード）*/
  outportb(0x6a, 1);           /*１６色モード*/
  setpal(0, 0, 0, 0);          /*パレット番号０＝黒*/
  setpal(6, 8, 15, 0);         /*パレット番号６＝茶色*/
  outportb(0xa2, 0x0d);        /*グラフィック画面　表示開始*/

  outportb(0x68, 0x0b);        /*ＫＣＧモード＝ビットマップモード*/
  kcod = 0x3020;
  for (i = 0; i < 25; i++) {
    for (j = 0; j < 40; j++) {
      vadd = i * 80 * 16 + j * 2;
      inpkan(kcod,kanpt);      /*文字パターンの読み込み*/
      for (k = 0; k < 16; k++) {
        kpat = kanpt[k * 2] + kanpt[k * 2 + 1] * 0x100;
        poke(0xa800, vadd, 0);        /* パレット番号６で書き込み*/
        poke(0xb000, vadd, kpat);
        poke(0xb800, vadd, kpat);
        poke(0xe000, vadd, 0);
        vadd += 80;
      }
```

```
      kcod++;
    }
  }
  outportb(0x68, 0x0a);       /*KCGモード=コードアクセスモード*/
}

void inpkan(int jiscod,char pbuf[32])    /*文字パターン読み出し関数*/
{
  int i, ploc;

  outportb(0xa1, jiscod % 0x100); /*文字コード下位バイト指定*/
  outportb(0xa3, jiscod / 0x100 - 0x20);       /*文字コード上位バイト指定*/
  for (i = 0; i < 32; i++) {
    ploc = i / 2;
    if ((i % 2) == 0) ploc |= 0x20;
    outportb(0xa5, ploc);     /*読み出し位置指定*/
    pbuf[i] = inportb(0xa9); /*パターンの読み出し*/
  }
}

void setpal(int pnum, int gbr, int rbr, int bbr)  /*パレット設定関数*/
{
  outportb(0xa8, pnum);
  outportb(0xaa, gbr);
  outportb(0xac, rbr);
  outportb(0xae, bbr);
}
```


■2-6-6 テキストBIOS

テキスト関係のBIOSは，INT 18Hによって呼び出されます．その機能は，大きく分けて，

1）画面モードの設定
2）カーソル制御
3）フォントパターンの読み出し・書き込み

の3つに分けられます．このうち，1）と2）については，同じ機能が簡単なI/O制御によって実現できてしまうため，BIOSコールを使っても，プログラムが簡単になるということは少なく，かえってI/O制御の方が簡単に済んでしまう場合が多いようです．では，これらのBIOSコールを使うとどのようなメリットがあるかというと，

・BIOSコールはワークエリアを書きかえるので，後で他のプログラムなどが現在の状況を知ることができる
・今後，98のハードが多少変更されたとしても，BIOSコールのしかたは変更されないと思われるので，長期的な互換性という点では有利になる
・多くのBIOSコールがノーマル・ハイレゾ共通であるため，両方のモードで動作するプログラムが組みやすい

などということがあります．結局，プログラムの安全性・互換性を取るならBIOSコール，簡便性・速度を取るならI/O直接制御ということになるでしょう．しかし，I/O直接制御でもそれほど不都合なことが起こるのはきわめてまれですから，一般にはI/O直接制御の方がよく行われているようです．そこで，以下に示すBIOSの解説では，そのBIOSコールとまったく同じことをI/O直接制御で簡単にできるときには，その方法も併記することにします．

なお，テキストBIOSはワークエリアを通して互いに関連しあっているので，ひとつでもテキスト関係のBIOSコールを用いる場合には，テキスト関係はすべてBIOSコールに統一した方がいいと思います．

それでは，以下にテキストBIOSの利用法を述べていきますが，注意すべきこととしては，以下のBIOSの説明に付けておいたサンプルプログラムのうちには，テキスト画面の表示形式を変えてしまうものがある，ということがあります．その場合，それ以後の操作に支障が出てくることがあるかもしれませんので，十分に注意してください．

■テキストBIOS一覧（INT 18H）

機能コード	機　　　能	ノーマル	ハイレゾ
０ＡＨ	テキスト画面モードの設定	○	○※
０ＢＨ	テキスト画面モードの取得	○	○※
０ＣＨ	テキスト画面の表示開始	○	○
０ＤＨ	テキスト画面の表示停止	○	○
０ＥＨ	テキスト画面表示開始アドレスの設定	○	○
０ＦＨ	テキスト画面への複数領域の設定	○	○
１０Ｈ	カーソル点滅の有無の設定	○	○
１１Ｈ	カーソルの表示開始	○	○
１２Ｈ	カーソルの表示停止	○	○
１３Ｈ	カーソルの表示位置の設定	○	○
１４Ｈ	フォントパターンの読み出し（１６ドット）	○	×
１６Ｈ	テキストＶＲＡＭのクリア	○	○
１ＡＨ	ユーザー定義文字の書き込み（１６ドット）	○	×
１ＢＨ	ＫＣＧアクセスモードの設定	○	○
１ＣＨ	テキスト関係のモード設定	×	○
１ＤＨ	表示幅の設定	×	○
１ＥＨ	カーソルタイプの設定	×	○
１ＦＨ	フォントパターンの読み出し（２４ドット）	×	○
２０Ｈ	ユーザー定義文字の書き込み（２４ドット）	×	○

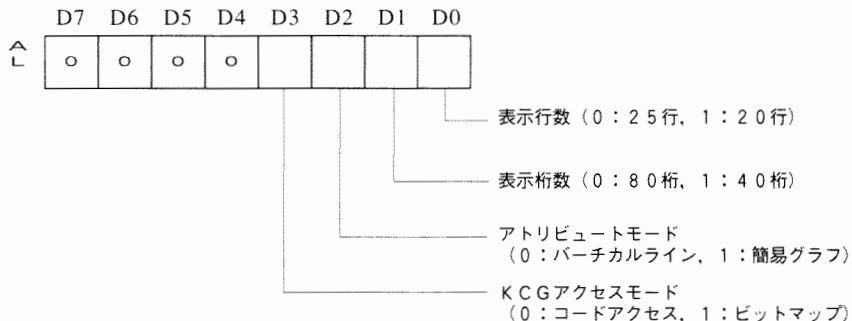
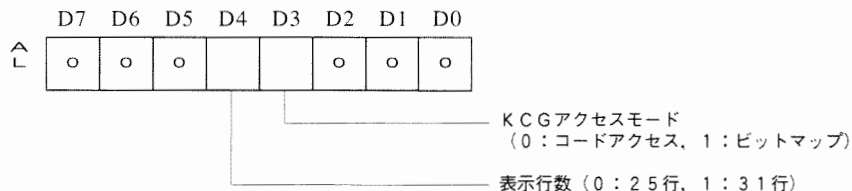
※　機能変更あり．

1 テキスト画面モードの設定

割り込み INT 18H

入　　力 AH←0AH（機能コード）

AL←画面モード指定コード

●ノーマルモード時

●ハイレゾモード時

出　　力 なし

解　　説 　GDCとモードフリップフロップをコントロールすることによって，テキスト画面のモード設定を行います．表示行数とは1画面に表示される行数，表示桁数とは1行あたりに表示される，半角文字で数えた文字数のことです．アトリビュートモードについては「2-6-2．テキスト」のI/Oの項を，KCGアクセスモードについては「2-6-5．KCGアクセス・ユーザー定義文字」の項を参照してください．

　このBIOSコールを用いて画面モードを設定するときには，機能コード0BHのBIOSコールによって現在の状態を取得し，必要な部分だけを変更して設定するのが無難です．また，このBIOSコールはカーソルの表示を停止してしまうので注意が必要です．

　なお，このBIOSコールの機能は，モードフリップフロップやGDCに値を出力することで容易に実現することができますが，その場合，ワークエリアが書きかえられないことに注意してください．

サンプル　（ノーマルモードのみ）

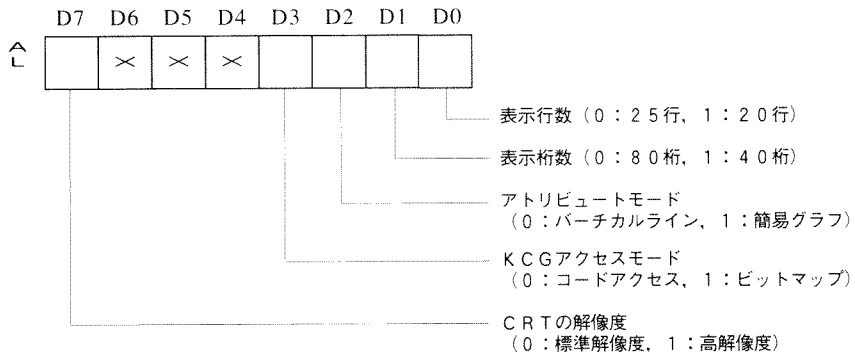
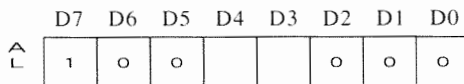
```
/* テキスト画面を８０桁×２５行モードに設定する */

#include <stdio.h>
#include <dos.h>

union REGS inregs, outregs;

void main(void)
{
    inregs.h.ah = 0x0b;                    /* 現在のモードの取得 */
    int86(0x18, &inregs, &outregs);
    inregs.h.al = outregs.h.al & 0xfc;     /* ビット０，１のクリア */
    inregs.h.ah = 0x0a;                    /* モードの設定 */
    int86(0x18, &inregs, &outregs);
}
```

2 テキスト画面モードの取得

割り込み　INT 18H

入　　力　AH←0BH（機能コード）

出　　力　AL←画面モードの状況

●ノーマルモード時

●ハイレゾモード時

KCGアクセスモード
（0：コードアクセス，1：ビットマップ）

表示行数（0：25行，1：31行）

解　　説　機能コード0AHのBIOSコールなどで設定された，テキスト画面の現在のモードを取得します．ここで読み出した値を，必要な部分だけ書きかえて機能コード0AHのBIOSコールでモード設定ができるようなデータ形式になっています．このBIOSコールで返される値は，現在ハードウェアに設定されている値ではなく，画面モード設定時にワークエリアに記録された値であるため，画面モードをI/O命令により直接制御するプログラムを実行した後では，このコールで得た値が正しいものである保証はありません．

サンプル　「1．テキスト画面モードの設定」のサンプル参照

3　テキスト画面の表示開始

割り込み　**INT 18H**

入　　力　**AH←0CH**（機能コード）

出　　力　なし

解　　説　テキストGDCにテキスト画面の表示開始を指示します．ワークエリアの書きかえなどを除けば，このコールの機能は次のようにするのと等価です（C言語の場合）．

```
outportb(0x62,0x0d);
```

サンプル

```
/* テキスト画面の表示を開始する */

#include <stdio.h>
#include <dos.h>

union REGS inregs, outregs;

void main(void)
{
    inregs.h.ah = 0x0c;                    /* 表示開始 */
    int86(0x18, &inregs, &outregs);
}
```

4 テキスト画面の表示停止

割り込み INT 18H

入　力 AH←0DH（機能コード）

出　力 なし

解　説 テキストGDCにテキスト画面の表示停止を指示します．ワークエリアの書きかえなどを除けば，このコールの機能は次のようにするのと等価です（C言語の場合）．

```
outportb(0x62,0x0c);
```

サンプル

```
/* テキスト画面の表示を停止する */

#include <stdio.h>
#include <dos.h>

union REGS inregs, outregs;

void main(void)
{
   inregs.h.ah = 0x0d;                    /* 表示停止 */
   int86(0x18, &inregs, &outregs);
}
```

5 テキスト画面表示開始アドレスの設定

割り込み INT 18H

入　力 AH←0EH（機能コード）

DX←テキスト画面の表示開始アドレス
（CPUから見たもの）

出　力 なし

解　説 テキスト画面の表示領域を1つとし，その表示開始アドレスを設定します．表示開始アドレスは，CPUから見たときのテキストVRAMの先頭アドレスからのオフセットアドレスで指定します（テキストVRAMの先頭が0000H）．ワークエリアの書きかえなどを除けば，通常このコールの機能は次のようにするのとほぼ等価です（C言語の場合）．

```
int   adr;
adr  =  (GDCから見た先頭アドレス);
outportb(0x62,0x70);
outportb(0x60,adr  %  0x100);
```

```
outportb(0x60,adr  /  0x100);

    ;  テキスト画面を裏画面表示にする
    ;
```

サンプル

```
/* テキスト画面を裏画面表示にする */

#include <stdio.h>
#include <dos.h>

union REGS inregs, outregs;

void main(void)
{
   inregs.x.dx = 0x1000;                    /* 裏画面の表示アドレス */
   inregs.h.ah = 0x0e;                      /* 表示開始アドレスの設定 */
   int86(0x18, &inregs, &outregs);
}
```

6 テキスト画面への複数領域の設定

割り込み INT 18H

入力 AH←0FH（機能コード）

BX←表示領域リストのセグメントアドレス

CX←表示領域リストのオフセットアドレス

DH←表示領域リストで設定し始める領域の番号（0～3）

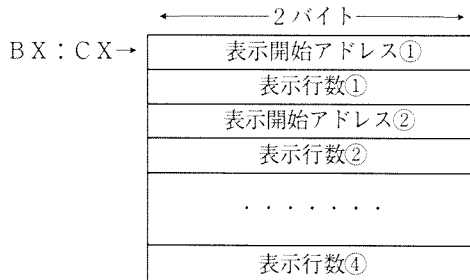
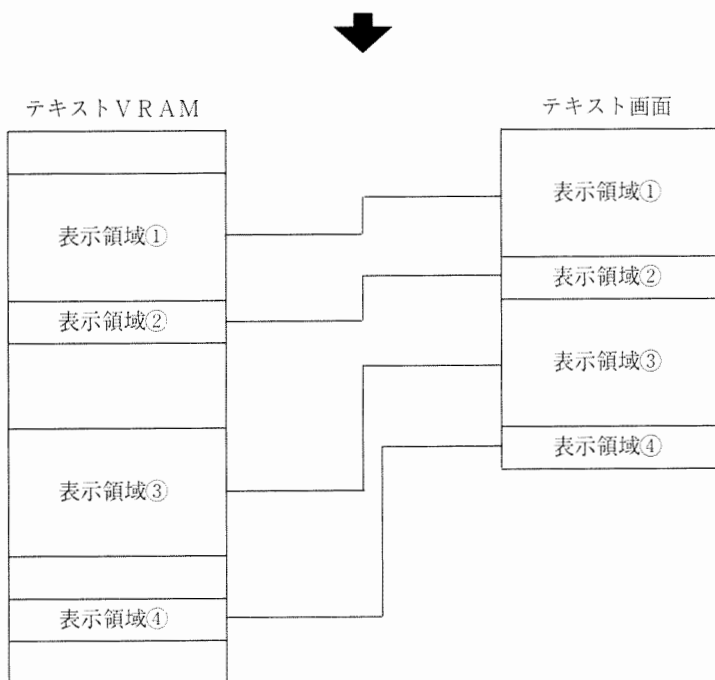
DL←表示領域リストで設定する領域の個数（1～4）

表示領域リスト←表示領域の情報（下図参照）

出力 なし

解説 テキスト画面を複数の領域に分割し，それぞれの領域の表示開始アドレスと表示行数を指定します．表示開始アドレスと表示行数は，BX：CXで示されるアドレスに指定しますが，その形式は下図のようにします．ここで指定する表示開始アドレスは，テキストVRAMの先頭番地を0000Hとした，CPUから見たときの相対アドレスです．

←2バイト→

BX：CX→
表示開始アドレス①
表示行数①
表示開始アドレス②
表示行数②
・・・・・・・
表示行数④

（DH＝0，DL＝4のとき）

サンプル

```
/* テキスト画面を上下 2 分割し，上半分を表画面，下半分を裏画面とする */

#include <stdio.h>
#include <dos.h>

union REGS inregs, outregs;

void main(void)
{
    int dlist[4] = {0x0000, 12, 0x1000, 13};  /* 表画面 1 2 行,
                                                   裏画面 1 3 行 */

    inregs.x.bx = FP_SEG(dlist);
    inregs.x.cx = FP_OFF(dlist);
    inregs.x.dx = 0x0002;                   /* 画面 2 分割 */
    inregs.h.ah = 0x0f;                     /* 表示開始アドレスの設定 */
    int86(0x18, &inregs, &outregs);
}
```

7 カーソル点滅の有無の設定

割り込み INT 18H

入　　力 AH←10H (機能コード)

AL←カーソル点滅の有無

AL=00H：カーソルを点滅させる

AL=01H：カーソルを点滅させない

出力 なし

解説 カーソルを点滅させるか否かを設定します．このBIOSコールは，自動的にカーソルの表示を停止するので，点滅の有無を設定したカーソルを表示させるためには，機能コード11HのBIOSコールを実行する必要があります．また，このBIOSコールは，GDCのCSRFORMコマンドによって点滅の有無を設定していますが，指定されたカーソルの点滅の有無の設定だけでなく，CSRFORMコマンドで設定する他の定数（1行のライン数，カーソルの点滅周期など）もすべてワークエリアに保存してある値で設定しなおします．したがって，I/O直接制御でこれらの値を変化させていた場合には，それらはすべて元の状態に戻されてしまうので注意してください．

サンプル

```
/* カーソルの点滅を停止する */

#include <stdio.h>
#include <dos.h>

union REGS inregs, outregs;

void main(void)
{
    inregs.h.al = 0x01;                    /* 点滅停止指示 */
    inregs.h.ah = 0x10;                    /* カーソル点滅の有無の設定 */
    int86(0x18, &inregs, &outregs);
    inregs.h.ah = 0x11;                    /* カーソルの表示開始 */
    int86(0x18, &inregs, &outregs);
}
```

8 カーソルの表示開始

割り込み INT 18H

入力 AH←11H（機能コード）

出力 なし

解説 テキストGDCにCSRFORMコマンドを出力することによって，カーソルの表示を開始します．このBIOSコールの機能は次のようにするのとほぼ等価です（C言語，25行モードの場合）．

```
outportb(0x62,0x4b);
outportb(0x60,0x8f);
```

サンプル　「7．カーソル点滅の有無の設定」のサンプル参照

9 カーソルの表示停止

割り込み　INT 18H

入　　力　AH←12H（機能コード）

出　　力　なし

解　　説　テキストGDCにCSRFORMコマンドを出力することによって，カーソルの表示を停止します．このBIOSコールの機能は次のようにするのとほぼ等価です（C言語，25行モードの場合）．

```
outportb(0x62,0x4b);
outportb(0x60,0x0f);
```

サンプル

```
/* カーソルの表示を停止する */

#include <stdio.h>
#include <dos.h>

union REGS inregs, outregs;

void main(void)
{
   inregs.h.ah = 0x12;                      /* カーソルの表示開始 */
   int86(0x18, &inregs, &outregs);
}
```

10 カーソル位置の設定

割り込み　INT 18H

入　　力　AH←13H（機能コード）

DX←カーソル位置をVRAM上のCPUアドレスで示した値

出　　力　なし

解　　説　テキストGDCにＣＳＲＷコマンドを出力することによって，カーソルの位置を設定します．カーソル位置はVRAM上のアドレスで指定しますが，これはCPUから見たアドレスで，VRAMの先頭番地を0000Hとしたオフセットアドレスで指定します．

サンプル

```
        ;  カーソルを約１秒間（４０，１２）の位置に設定する
```

```
/* カーソルを約1秒間（40，12）の位置に設定する */

#include <stdio.h>
#include <dos.h>

union REGS inregs, outregs;

void main(void)
{
    int i;

    inregs.x.dx = 12 * 160 + 40 * 2;      /* カーソルのCPUアドレス */
    inregs.h.ah = 0x13;                   /* カーソル位置の設定 */
    int86(0x18, &inregs, &outregs);
    for (i = 1; i <= 60; i++) {           /* 時間待ち（VSYNC60回分） */
        while ((inportb(0x60) & 0x20) != 0);
        while ((inportb(0x60) & 0x20) == 0);
    }
}
```

11 フォントパターンの読み出し（16ドット） ノーマル

割り込み INT 18H

入力 AH←14H（機能コード）

BX←フォントパターンバッファのセグメントアドレス

CX←フォントパターンバッファのオフセットアドレス

DX←パターンを読み出す文字の文字コード

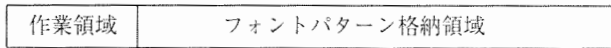
出力 フォントパターンバッファ←フォントパターン

解説 漢字，ANK文字などのフォントパターン（文字パターン）を読み出します．文字の種類によって，どのような文字コードを指定し，パターンバッファはどれほど確保すればよいかをまとめると，次のようになります．

文字の種類	指定する文字コード		文字のサイズ（横×縦ドット）	パターンバッファの必要サイズ（バイト）
	DH	DL		
ANK文字	80H	ANK	8×16	18
全角漢字	JIS漢字コード		16×16	34
半角漢字	JIS漢字コード		8×16	18
1／4角文字	00H	ANK	8×8	10

ANK：ANK文字のキャラクタコード

また，フォントパターンバッファに格納されるパターンデータの形式は次のようなものです．

←2バイト→←（横のドット数／8×縦のドット数）バイト→

作業領域	フォントパターン格納領域

フォントパターン格納領域のサイズは，たとえばANK文字なら16バイト，全角漢字なら32バイトになります．フォントパターン格納領域には，フォントパターンの上の方のデータから順番に格納されていきます．全角文字の場合には，先頭に左上のパターンが格納され，次に右上，上から2番目の左，上から2番目の右・・・と順次格納されます．つまり，全角文字については，CGウィンドウでのデータ形式と同じ形式になっています．

サンプル　（ノーマルモードのみ）

```
/* '技'の字をテキスト画面に拡大表示する */

#include <stdio.h>
#include <dos.h>

union REGS inregs, outregs;

void main(void)
{
   int i, j;
   unsigned vadd, bwin, patt;
   unsigned char patbuf[17 * 2];

   inregs.x.bx = FP_SEG(patbuf);
   inregs.x.cx = FP_OFF(patbuf);
   inregs.x.dx = 0x353b;                 /* '技'の字のＪＩＳコード */
   inregs.h.ah = 0x14;                   /* 文字パターンの読み出し */
   int86(0x18, &inregs, &outregs);
   vadd = 160 * 4 + 32 * 2;              /* ４行目の３２桁目から */
   for (i = 1; i <= 16; i++) {
      patt = patbuf[i * 2] * 0x100 + patbuf[i * 2 + 1];
      bwin = 0x8000;                     /* ビットの窓 */
      for ( j = 1; j <= 16; j++) {
         if ((patt & bwin) != 0) poke(0xa000, vadd, 0x0087);
                               else poke(0xa000, vadd, 0x0020);
         bwin = bwin >> 1;
         vadd += 2;
      }
      vadd += 64 * 2;
   }
}
```

12 テキストVRAMのクリア

割り込み INT 18H

入　力 AH←16H（機能コード）

DH←アトリビュートエリアを埋めるアトリビュートデータ

DL←文字エリアを埋めるANK文字コード

出　力 なし

解　説 テキストVRAMの全面を，DL，DHで指定する文字・アトリビュートで埋めます．DH=E1H，DL=20Hを指定すると，ごく普通のテキスト画面のクリアになります．

サンプル

```
/* テキスト画面をクリアする */

#include <stdio.h>
#include <dos.h>

union REGS inregs, outregs;

void main(void)
{
    inregs.x.dx = 0xe120;                /* 白のスペースでクリア指定 */
    inregs.h.ah = 0x16;                  /* テキストＶＲＡＭのクリア */
    int86(0x18, &inregs, &outregs);
}
```

13 ユーザー定義文字の書き込み（16ドット） ノーマル

割り込み INT 18H

入　力 AH←1AH（機能コード）

BX←フォントパターンバッファのセグメントアドレス

CX←フォントパターンバッファのオフセットアドレス

DX←登録するユーザー定義文字の文字コード（JISコード）

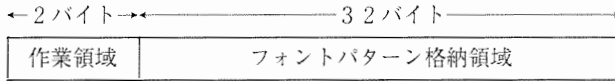
フォントパターンバッファ←フォントパターン

出　力 なし

解　説

指定された文字コードのユーザー定義文字に，フォントパターンバッファに指定された文字パターンを書き込みます．ユーザー定義文字の文字コードは，PC-9801E/F/MではJISコードの7621H～765FHの63文字，それ以降の機種では7621H～767EH,および7721H～777EHの188文字です．

フォントパターンバッファに格納すべきパターンデータの形式は次のようなものです．

←2バイト→	←32バイト→
作業領域	フォントパターン格納領域

このうち，ユーザーがデータをセットしておかなければならないのはフォントパターン格納領域で，ここには設定するフォントパターンを，パターンの左上，右上，左の上から2番目，右の上から2番目・・・というようにして（つまり，フォントパターンの読み出しのときと同じデータ形式で），合計32バイトのパターンデータをセットしておきます．

サンプル

（ノーマルモードのみ）

```
/* テキスト画面を網かけ模様で埋める */

#include <stdio.h>
#include <dos.h>

union REGS inregs, outregs;

void main(void)
{
   int i;
   unsigned patbuf[17];

   for (i = 1; i <= 16; i+= 2) {          /* 網掛け模様セット */
      patbuf[i    ] = 0x1111;
      patbuf[i + 1] = 0x4444;
   }
   inregs.x.bx = FP_SEG(patbuf);
   inregs.x.cx = FP_OFF(patbuf);
   inregs.x.dx = 0x7621;                  /* JISコード7621H */
   inregs.h.ah = 0x1a;                    /* ユーザー定義文字の書き込み */
   int86(0x18, &inregs, &outregs);
   for (i = 0; i < 160 * 24; i += 2) {     /* VRAMへの書き込み */
      poke(0xa000, i, 0x2156);            /* 7621Hの文字 */
   }
}
```

14 KCGアクセスモードの設定

割り込み INT 18H

入　力 AH←1BH（機能コード）

AL←KCGアクセスモード指定コード（00H：コードアクセス，
01H：ビットマップ）

出　力 なし

解　説

KCGアクセスモードの設定を行います．KCGアクセスモードの詳細については，「2-6-5．KCGアクセス・ユーザー定義文字」の項を参照してください．このKCGアクセスモードは，機能コード0AHのBIOSコールでも変更することができ，また，I/O直接制御でも容易に変更することができます．ワークエリアの書きかえなどを除けば，このBIOSコールの機能は次のようにするのと等価です（C言語の場合）．

コードアクセスモードにするには，

```
outportb(0x68,0x0a);
```

ビットマップモードにするには；

```
outportb(0x68,0x0b);
```

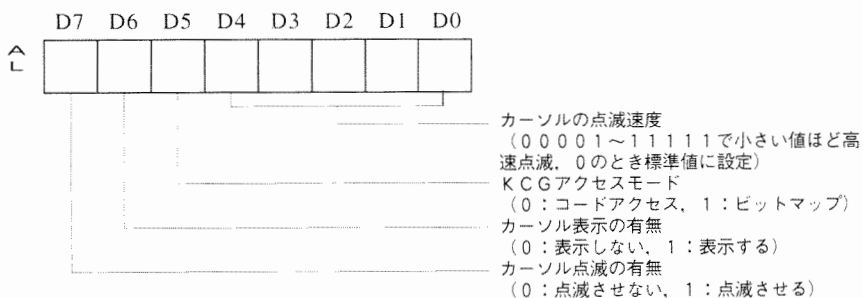
サンプル

```
/* ＫＣＧアクセスモードをビットマップにする */

#include <stdio.h>
#include <dos.h>

union REGS inregs, outregs;

void main(void)
{
    inregs.h.al = 0x01;                   /* ビットマップモード指定 */
    inregs.h.ah = 0x1b;                   /* ＫＣＧアクセスモードの設定 */
    int86(0x18, &inregs, &outregs);
}
```

15 テキスト関係のモード設定　ハイレゾ

割り込み INT 18H

入　力 AH←1CH（機能コード）

AL←モード指定コード

DH←カーソル表示開始ライン　（00H～0EH）

DL←カーソル表示終了ライン　（00H～0EH）

出　力　なし

解　説　テキスト画面関係のモード設定，主にカーソル関係のものの設定を行います．カーソルの点滅速度，カーソル表示開始ライン・終了ラインなどについては，テキストGDCのCSRFORMコマンドの解説の部分を参照してください．

サンプル　（ハイレゾモードのみ）

```
/* カーソルを点滅なしのアンダーラインカーソルにする */

#include <stdio.h>
#include <dos.h>

union REGS inregs, outregs;

void main(void)

{
    inregs.h.al = 0x4c;                    /* 点滅なし指定 */
    inregs.x.dx = 0x0e0e;                  /* アンダーラインカーソル指定 */
    inregs.h.ah = 0x1c;                    /* モードの設定 */
    int86(0x18, &inregs, &outregs);
}
```

16 表示幅の設定 ハイレゾ

割り込み　**INT 18H**

入　力　**AH←1DH**（機能コード）

AL←VRAMの横幅の文字数

出　力　なし

解説 テキストVRAMの横幅を，文字数で指定します．ALに指定する値は，GDCのPITCHコマンドで指定するパラメータと同じ意味で，通常の値は８０です．

サンプル （ハイレゾモードのみ）

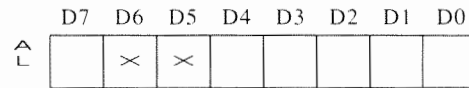
```
/* テキスト画面の表示を１行おきにする */

#include <stdio.h>
#include <dos.h>

union REGS inregs, outregs;

void main(void)
{
    inregs.h.al = 160;                  /* ＶＲＡＭの横幅＝通常の２倍 */
    inregs.h.ah = 0x1d;                 /* 表示幅の設定 */
    int86(0x18, &inregs, &outregs);
}
```

17 カーソルタイプの設定 ハイレゾ

割り込み INT 18H

入力 AH←1EH（機能コード）

AL←カーソルタイプ指定コード

D7	D6	D5	D4	D3	D2	D1	D0
	×	×					

D4〜D0：カーソルの点滅速度（０００００１〜１１１１１で小さい値ほど高速点滅，０のとき標準値に設定）

D7：カーソル点滅の有無（０：点滅させない，１：点滅させる）

DH←カーソル表示開始ライン（00H〜0EH）

DL←カーソル表示終了ライン（00H〜0EH）

出力 なし

解説 カーソルの属性を設定します．各設定値の意味は機能コード1CHのBIOSコールのものと同じです．

サンプル （ハイレゾモードのみ）

```
/* カーソルを点滅なしのブロックカーソルにする */

#include <stdio.h>
#include <dos.h>
```

```
union REGS inregs, outregs;

void main(void)
{
   inregs.h.al = 0x0c;                    /* 点滅なし指定 */
   inregs.x.dx = 0x000e;                  /* ブロックカーソル指定 */
   inregs.h.ah = 0x1e;                    /* カーソルタイプの設定 */
   int86(0x18, &inregs, &outregs);
}
```

18 フォントパターンの読み出し(24ドット) ハイレゾ

割り込み INT 18H

入　力 AH←1FH (機能コード)

DS←フォントパターンバッファのセグメントアドレス

BX←フォントパターンバッファのオフセットアドレス

DX←読み出すパターンの文字コード

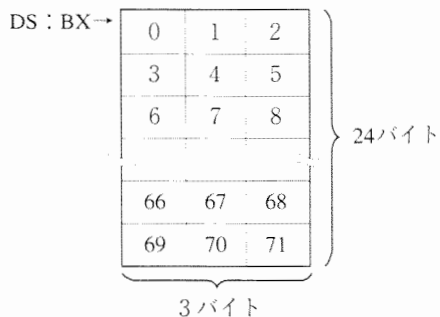
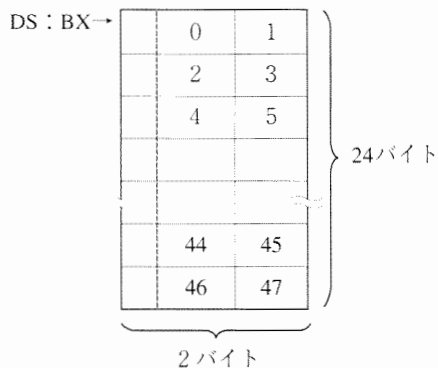
出　力 フォントパターンバッファ←フォントパターン

解　説 漢字，ANK文字などのフォントパターン（文字パターン）を読み出します．文字の種類によって，どのような文字コードを指定し，パターンバッファはどれほど確保すればよいかをまとめると，次のようになります．

文字の種類	指定する文字コード		文字のサイズ(横×縦ドット)	パターンバッファの必要サイズ(バイト)
	DH	DL		
ANK文字	00H	ANK	14×16	48
全角漢字	JIS漢字コード		24×24	72
半角漢字	JIS漢字コード		14×24	48

ANK：ANK文字のキャラクタコード

ノーマルモードのBIOSコールと異なり，このBIOSコールではフォントパターンバッファの中に作業領域は取られず，パターンバッファの先頭からパターンが格納されていきます．その格納形式は次のようなものです．

① 日本語（全角）

② 日本語（半角），ANK

サンプル （ハイレゾモードのみ）

```
/* '技'の字をグラフィック画面に表示する */

#include <stdio.h>
#include <dos.h>

union REGS inregs, outregs;
struct SREGS segregs;

void main(void)
{
   int i, j, vadd;
   char patbuf[72];                       /* ７２バイトのパターンバッファ */

   outportb(0xa2, 0x0d);                  /* グラフィック画面表示開始 */
   outportb(0xa4, 0x00);                  /* 描画対象＝全プレーン */
   segregs.ds = FP_SEG(patbuf);
   inregs.x.bx = FP_OFF(patbuf);
   inregs.x.dx = 0x353b;                  /* '技'の字のＪＩＳコード */
   inregs.h.ah = 0x1f;                    /* 文字パターンの読み出し */
   int86(0x18, &inregs, &outregs);
   vadd = 0;
   for (i = 0; i < 24; i++) {
      for (j = 0; j < 3; j++) {
         pokeb(0xc000, vadd, patbuf[i * 3 + j]);
         vadd++;
      }
      vadd += 137;
   }
}
```

19 ユーザー定義文字の書き込み(24ドット) ハイレゾ

割り込み INT 18H

入　力 AH←20H (機能コード)

DS←フォントパターンバッファのセグメントアドレス

BX←フォントパターンバッファのオフセットアドレス

DX←登録するユーザー定義文字の文字コード (JISコード)

フォントパターンバッファ←フォントパターン

出　力 なし

解　説 指定された文字コードのユーザー定義文字に，フォントパターンバッファに指定された文字パターンを書き込みます．ユーザー定義文字の文字コードは，JISコードの7621H～765FHの63文字です．

フォントパターンバッファに格納すべきパターンデータの形式については，フォントパターンの読み出し（機能コード1FHのBIOSコール）の全角文字の形式と同じなので，そちらを参照してください．

サンプル （ハイレゾモードのみ）

```
/* テキスト画面を網かけ模様で埋める */

#include <stdio.h>
#include <dos.h>

union REGS inregs, outregs;
struct SREGS segregs;

void main(void)
{
   int i;
   char patbuf[72];                     /* 72バイトのパターンバッファ */

   for (i = 0; i < 72; i++) {           /* 網掛け模様 */
      if ((i % 6) < 3) patbuf[i] = 0x11;
                  else patbuf[i] = 0x44;
   }
   segregs.ds = FP_SEG(patbuf);
   inregs.x.bx = FP_OFF(patbuf);
   inregs.x.dx = 0x7621;                /* JISコード7621H */
   inregs.h.ah = 0x20;                  /* ユーザー定義文字の書き込み */
   int86(0x18, &inregs, &outregs);
   for (i = 0; i < 160 * 24; i += 2) {
         poke(0xe000, i, 0x2156);       /* 7621Hの文字 */
   }
}
```

§2-7 グラフィック

グラフィックというのは，画面上に一般的な図形や絵を表示するためのものです．前に述べたテキスト画面には，ある程度決まったパターン（文字）しか表示できませんでしたが，グラフィック画面には，解像度が許すかぎりあらゆるパターンを表示することが可能で，テキスト画面とは比較にならないくらいきれいな画面表示を実現することができます．

コンピュータの画面表示は，画面をよく見るとわかりますが，細かい無数の点によって作られています．98の画面は，ノーマルモードでは通常，横方向は640個，縦方向は400個の点に分割されています．つまり，画面の構成単位である1つの点は，横は画面の横の大きさの1／640，縦は画面の縦の大きさの1／400の大きさということになります．この，画面の構成単位となる点のことをドットといいます．98ではこのドットの大きさの縦横の比は1：1になっており，したがって画面の大きさの縦横比の方も横640対縦400になっています．

コンピュータは，これらのそれぞれのドットを光らせるか光らせないか，どんな色でどのくらいの明るさで光らせるかを制御することによって，文字や図形の表示を実現しています．上に述べた横方向のドット数640，縦方向のドット数400（以下，640×400ドットと略）のときには，画面上に表示されているドットの総数は，640×400＝256,000ドットとなります．グラフィック画面では，この膨大な数の点のすべてに対して，点があるかないかを指定し，あるいはすべての点の色を指定します．そのため，ドットがあるかないかだけの単色表示をするとしても，1ドットを表すのに1ビットは必要ですから，256,000ビット＝32,000バイト（32KB）のデータが必要になることになります．さらに，たとえば16色のカラー表示をしようとすると，1ドットに関しての情報量はこの4倍（$\log_2 16$倍）になりますから，全体のデータ量もこの4倍の128KBが必要になります．テキスト画面を表示するのに必要なデータの量が色指定を含めてもせいぜい6KBであることを考えると，グラフィック画面の表示にいかに膨大な量のデータが必要であるかがわかっていただけると思います．

このデータ量の膨大さのために，グラフィック画面の扱いには常に困難がつきまといます．静止画を表示するときでも，1画面全体ではメインメモリの1／5，2HDのディスク1枚の1／10にも達する多量のデータが必要となります．さらに，動画を表示しようとすると，書き込むべきデータ量の多さから速度の問題も大問題になります．しかし，グラフィックはいってみればコンピュータの「花形」で，ゲームにも，業務用ソフトにも，最近ではWindowsなどのOSにもなくてはならないものです．そこで，以下に，いかに効率よくグラフィックを使うかなどといったことも含めて，グラフィック関係のハード・BIOSや，その具体的な利用方法について述べていきます．

■2-7-1 画面モード

98のグラフィック表示には，解像度や同時に表示できる色の数が異なるいくつかの画面モードがあります．以下に，98が持っているグラフィック画面モードを列挙し，それぞれのモードの特徴について述べていきます．

■解像度モード

解像度モードは，グラフィック画面の解像度を規定します．解像度が高ければ画面はきれいになりますが，必要なデータの量は多くなります．98の解像度モードには，ノーマルモードでは基本的に2つのモードがあります．これらのモードを切り替えるには，BIOSコールと，GDCおよびモードフリップフロップを直接コントロールするという2つの方法がありますが，詳しくは関係各項を参照してください．

1）640×400ドットモード（400ラインモード）

640×400ドットモードは，ノーマル98に可能な最高解像度であり，このモードのときは，横方向のドット数が640，縦方向のドット数が400です．画面表示では，縦方向の1ドットのことを1ラインとも呼ぶので，このモードは400ラインモードとも呼ばれます．このモードでは，次に述べる640×200ドットモードに比べ緻密な画像を表示することができますが，1画面を表示するのに640×200ドットモードの2倍のデータが必要なので，データ量や速度の面では不利になります．

2）640×200ドットモード（200ラインモード）

640×200ドットモードは，主にPC-8801シリーズとの互換性を保つために用意されたもので，このモードのときは，横方向のドット数が640，縦方向のドット数が200です．ノーマル98に可能な最高解像度である640×400ドットモードに比べ，縦方向の解像度が1／2で，若干ドットの荒さが目立ちますが，1画面を表示するのに必要なデータの量が640×400ドットモードの1／2で済むため，動画表示が必要なアクションゲームなどでよくこのモードが用いられます．

ただし，この640×200ドットモードはGDC 2.5MHzモードでのみ実現可能であり，GDC 5MHzモードでは使用できません．

なお，ハイレゾモードでは，解像度モードは基本的に1120×768ドットに固定されています．

■カラーモード

カラーモードは，画面上に同時に表示できる色の数を規定します．当然のことながら，同時に表示できる色数を多くすれば必要なデータ量も多くなります．98のノーマルモードでは基本的に3つのカラーモードがあります．これらを切り替えるには，LIOおよびBIOSのコールと，モードフリップフロップの直接制御の2つの方法がありますが，詳しくは関係各項を参照してください．

なお，ハイレゾモードでは，カラーモードは基本的に16色モードに固定されています．

1）16色モード

16色モードは，4096色中16色を表示できるモードです．これは若干わかりにくいかもしれませんが，

次のような意味です．グラフィック画面のドットのそれぞれには，個別にパレット番号という番号を指定でき，各ドットの色はその指定したパレット番号によって決まります．16色モードでは各ドットについて指定できるパレット番号が0～15までの16種類で，その0～15までのパレット番号に，4096色の中から任意の色を「パレット」として割り当てることができるのです．結局，画面に同時に表示できる色は16色ですが，その表示する16色は4096色の中から自由に選ぶことができるのです．

なぜ，4096色中からの選択なのかというと，16色モードでは各パレット番号に，光の3原色であるG（Green，緑），R（Red，赤），B（Blue，青）のそれぞれについて，0～15までの16階調の明るさを持った色を割り当てることができるからです．たとえば，G＝15，R＝0，B＝0とすればそのパレット番号の色は純粋な緑になり，G＝15，R＝8，B＝0とすれば黄緑色になります．このように，3つの原色について独立に16階調が指定できるわけですから，結果的に，16^3＝4096色の中から任意の色を選ぶことができることになるわけです．

このモードでは，16種類のパレット番号をドットごとに指定するのですから，1ドット当たりに必要なビット数は4ビット（$\log_2 16$ビット）になります．したがって，画面を表示するのに必要なデータ量は400ラインモードのときは128KB，200ラインモードのときは64KBとなります．

なお，この16色モードは，98のノーマルモードでは最も一般的なモードで，パレットをうまく組み合わせることで，以下に述べる8色モードやモノクロモードをほとんど含むことができます．したがって，16色モードが搭載されている機種では，グラフィック表示にはたいていこの16色モードを使います．

2）8色モード

8色モードは，黒，青，赤，紫，緑，水色，黄色，白の8色の中から任意の8色を選んで表示することができるモードです．8色の中から8色を選ぶというのは無意味なように思えますが，そうではありません．8色モードでは各ドットについて指定できるパレット番号は0～7までの8種類で，その0～7までのパレット番号に，上に挙げた任意の色を「パレット」として割り当てることができます．たとえば，その気になれば0～7までのパレット番号すべてに白を割り当てることも可能です．もっとも，そんなことをすれば画面に何が書いてあっても画面全部が真っ白に表示されてしまうので，あまり意味がありませんが．通常は，上に示した色の順番に0～7に割り当てられています．

このモードでは8種類のパレット番号をドットごとに指定するため，1ドットにつき3ビット（$\log_2 8$ビット）が必要となります．そのため，400ラインモードでは1画面の表示に96KBのデータが，200ラインモードでは48KBのデータが必要になります．

なお，BASICのマニュアルなどを見ると，4096色中8色モードというものがありますが，これはハードウェア的には16色モードとまったく同じものです．16色モードが搭載されている機種では，普通，8色しか使わなくても8色モードは使わず，16色モードを使います．詳しくは16色モードの項目を参照してください．

3）モノクロモード

モノクロモードは，2色（通常は，白と黒）しか必要ないときのためのモードです．1画面の表示に必要なデータの量は，400ラインモードのとき32KB，200ラインモードのとき16KBです．グラフィックモードの中では1番少ないデータ量で表示ができるので，画面の見映えより速度が必要な処理に適しています．

モノクロモードといってもまったく色が付けられないわけではなく，このモードでは，グラフィック画面の色もテキスト画面のアトリビュートエリアに指定された色で表示されます．したがって，モノクロモードでもキャラクタ単位（8×16ドット単位）で色を付けることは可能です．が，やはり基本的には白黒表示しかできないので，普通はこのモードのことをカラーモードとは呼びません．

このモノクロモードには，実は2つの種類があります．1つは8色モードをモノクロモードにしたもの，もう1つは16色モードをモノクロモードにしたものです．モノクロモードというのは基本的に，各カラーモードでのドットの色や明るさなどをすべてドットがあるかないかだけに置きかえてしまい，それにアトリビュートエリアに指定された色を付けるものなのです．8色モードのときはドットの色のGRBのうち1つでも1のものがあるとき，16色モードではドットの色のGRBのうち1つでも輝度が8を超えるものがあるときに，そのドットはあるものと見なされ，アトリビュートエリアで指定された色で画面に表示されます．

なお，上に挙げた解像度モードとカラーモードは，それぞれ独立に指定することができます．たとえば，「640×400ドット・16色モード」，「640×200ドット・モノクロモード」といったように両者を自由に組み合わせて使うことが可能です．

■サンプルプログラム（ノーマルモードのみ）

このサンプルプログラムは，プログラムの最初の部分で描いた同じ図形を，さまざまなカラーモード・解像度で表示するプログラムです．98のカラーモードや解像度の決定のしかたを明確に示すために，モード設定はすべてI／O直接制御によって行っています．

このプログラムを少し変更して，たとえばGDCは200ラインモードのときの設定，モードF／F1は400ラインモードのときの設定にすると，普通の200ラインモードとは違う200ラインモードを実現することもできます．実際に試してみて下さい．

ただし，GDCが5MHzモードになっていると，このプログラムでは200ラインモードを実現することはできません．きちんと200ラインモードで表示したいときは，GDCを2.5MHzに設定する必要があります．

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>

void main(void)
{
    int i, x, y, cl, addr;

    outportb(0xa2, 0xd);                    /* グラフィック表示開始 */
    outportb(0x6a, 1);                      /* 16色モード */
                                            /* 傾斜図形の書き込み */
    for (y = 16; y < 400; y++) {            /* y = 16〜399 */
        for (x = 0; x < (640/8) ; x++) {/* x = 0〜639(1バイト,8ドット単位) */
            cl = (x + y) % 16;              /* 傾斜図形の色 */
            addr = y * 80 + x;              /* VRAMアドレス */
                                            /*以降, 1バイト(8ドット)分の書き込み*/
            if ((cl & 1) != 0) pokeb(0xa800, addr, 0xff);
                              else pokeb(0xa800, addr, 0x00);
```

```
            if ((cl & 2) != 0) pokeb(0xb000, addr, 0xff);
                                 else pokeb(0xb000, addr, 0x00);
            if ((cl & 4) != 0) pokeb(0xb800, addr, 0xff);
                                 else pokeb(0xb800, addr, 0x00);
            if ((cl & 8) != 0) pokeb(0xe000, addr, 0xff);
                                 else pokeb(0xe000, addr, 0x00);
        }
    }
    for (addr = 0; addr < (80 * 16); addr += 2) {
        poke(0xa800, addr, 0);                   /* 0〜15ラインのクリア */
        poke(0xb000, addr, 0);
        poke(0xb800, addr, 0);
        poke(0xe000, addr, 0);
    }

    for (i = 0; i < 16; i++) {
        outportb(0xa8, i);                       /* 16色モードのパレット設定 */
        outportb(0xaa, i);
        outportb(0xac, 0);
        outportb(0xae, i);
    }

    clrscr();                                    /* 640×400ドット・16色モード */
    printf("640×400ドット・16色モード");
    outportb(0x6a, 1);                           /* 16色モード */
    outportb(0xa2, 0x4b);                        /* GDC CSRFORMコマンド*/
    outportb(0xa0, 0);                           /* L/R = 0 */
    outportb(0x68, 8);                           /* モードF/F1 */
    getchar();

    clrscr();                                    /* 640×200ドット・16色モード */
    printf("640×200ドット・16色モード");
    outportb(0x6a, 1);                           /* 16色モード */
    outportb(0xa2, 0x4b);                        /* GDC CSRFORMコマンド */
    outportb(0xa0, 1);                           /* L/R = 1 */
    outportb(0x68, 9);                           /* モードF/F1 */
    getchar();

    clrscr();                                    /* 640×400ドット・8色モード */
    printf("640×400ドット・8色モード ");
    outportb(0x6a, 0);                           /* 8êFモード */
    outportb(0xa2, 0x4b);                        /* GDC CSRFORMコマンド*/
    outportb(0xa0, 0);                           /* L/R = 0 */
    outportb(0x68, 8);                           /* モードF/F1 */
    getchar();

    clrscr();                                    /* 640×200ドット・8色モード */
    printf("640×200ドット・8色モード");
    outportb(0x6a, 0);                           /* 8色モード */
    outportb(0xa2, 0x4b);                        /* GDC CSRFORMコマンド */
    outportb(0xa0, 1);                           /* L/R = 1 */
    outportb(0x68, 9);                           /* モード F/F1 */
    getchar();
```

グラフィックVRAMは，画面上のドットすべてについての情報を記憶しているRAM領域です．98の最高解像度である640×400ドットモードでは，画面いっぱいの単色表示をするのに32KBのデータが必要ですが，98では，この32KBの全画面・単色表示のデータを1まとまりとしてプレーンと呼んでいます．モノクロ表示のときには1つのプレーンが1枚の画面を作り，カラーモードでは複数のプレーンを重ね合わせることによって，1枚の画面を構成します．98では，このプレーンが，4プレーン1組のものが2組存在していて，合計32KB×8＝256KBのVRAMが存在しています．そして，それら2組のプレーン群はそれぞれ表画面，裏画面と呼ばれています．

実際のプレーンのデータ形式を図2-30に示します．98には，同じ形式のプレーンがメモリアドレスA8000H，B0000H，B8000H，E0000Hの4ヵ所に存在しています．以下，これらをそれぞれプレーン0，1，2，3と呼ぶことにします．また，これらと同じアドレスにそれぞれもう1枚ずつ，裏画面のプレーンが存在しています．表画面のプレーンと裏画面のプレーンを同時に使うことはできず，両者は切り替えて使うことになります（ただし，表画面を表示しつつ裏画面に書き込みを行うといったことは可能）．両者の切り替えは，I／Oポート A4Hおよび A6Hを制御することによって行います．詳しくは「2−7−3．グラフィックのI／O」を参照してください．これら表裏画面の4枚のプレーンがどのように使われるかは，カラーモードによって異なるので，以下にそれぞれのカラーモードについて，各プレーンがどのように使われるかを述べていきます．

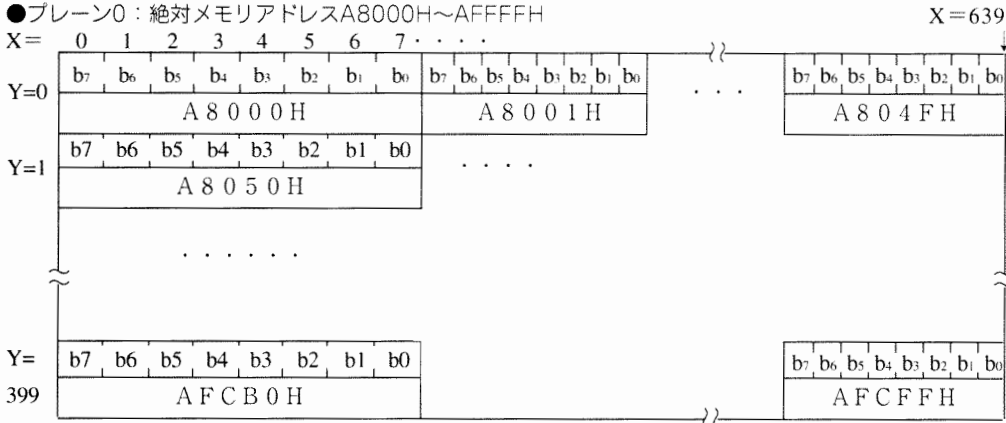



図2-30　プレーンのデータ形式

1）16色モード

16色モードでは，プレーン0～プレーン3までの4プレーンすべてを使い，これら4プレーンを重ね合わせることによって1つの画面を構成します．それぞれのプレーンの意味付けは，パレットが変更されていなければ，プレーン0が光の三原色（青，赤，緑）のうちの青，プレーン1が赤，プレーン2が緑を表し，プレーン3は色のインテンシティ（輝度）を表しています．

つまり，画面上のあるドットに注目したとき，プレーンnのそのドットに相当する部分のビットを2^nの桁に持つような数が，前に述べたそのドットのパレット番号となりますが（たとえばプレーン0のビットが0，プレーン1が1，プレーン2が0，プレーン3が1なら1010B＝0AH），今はプレーンが4枚ですから各ドットのパレット番号は0～15（2進数の0000B～1111B）の数で，2^0の桁が青を，2^1の桁が赤を，2^2の桁が緑を，2^3の桁がインテンシティを表すことになります．もっとも，この0～15の各パレット番号に割り当てる色は，パレットを指定することによって4096色中から任意に指定することができます．

このように16色モードのときには，1画面を構成するのに4プレーンが必要です．したがって，切り替えて使用できる画面数は，400ラインモードのときは表画面，裏画面の2枚です．200ラインモードのときは，1画面を作るのにVRAMの半分しか必要ありませんから，表裏画面のそれぞれについてVRAMの前半，後半と分けて使うことができ，全体で4枚の画面を切り替えて使え，4枚で繰り返すアニメーションなどを実現することができます．

2）8色カラーモード

8色カラーモードでは，プレーン3はメモリ上に存在せず，有効なプレーンは1画面当たりプレーン0～2の3プレーンで，これら3プレーンを重ね合わせることによって1つの画面を構成します．それぞれのプレーンの意味付けは，パレットが変更されていなければ，プレーン0が光の三原色の青，プレーン1が赤，プレーン2が緑を表しています．したがって，あるドットのパレット番号が0～7のとき，そのドットの色はそれぞれ黒，青，赤，紫，緑，水色，黄色，白になります．

これが，8色カラーモードにおける普通の発色のしかたですが，実際には，上に示したドットのパレット番号と色の関係は固定されておらず，0～7の任意のパレット番号に上に挙げた8色のうちの任意の色を割り当てることができます．

このように，8色カラーモードでは1画面を構成するのに3プレーンを必要とするため，切り替えて使用できる画面数は16色モードのときと同じで，400ラインモードのとき表裏の2枚，200ラインモードのとき表前半，表後半，裏前半，裏後半の4枚です．

3）モノクロモード

モノクロモードでは，各プレーンは原則的にそれぞれ独立の画面として扱われ，1プレーンで1枚のグラフィック画面を構成します．したがって，400ラインモードでは，表裏画面合わせて8つのプレーンを使って，8枚の画面を設定して，それらを切り替え表示することができます．200ラインモードでは1プレーンで2枚の画面を構成できるので，さらに多く，16枚の画面を切り替えて表示することができます．画面の切り替えは，表裏画面の切り替えとパレット制御，それに200ラインモードのときはGDCによる表示開始アドレスの変更も組み合わせて行います．

なお，グラフィックVRAMを扱う時には，次のようなことに注意することが必要です．

1) グラフィックVRAMは一般のRAMよりも低速である
2) グラフィックVRAMは，たとえ32ビットマシンでも16ビットバスを通してつながっている

1) は，ハードウェア的には，VRAMアクセス時には多くのウェイトがかかるということを意味します．このため，VRAMに1バイト書き込むには，メインRAMに1バイト書き込むのにかかる時間の3〜7倍（機種によって異なる）もかかってしまいます．このことは，ただでさえ大量のデータを書き込まなければならないグラフィック処理をさらに遅くする要因になってしまっています．

そして2) は，低速なVRAMへのアクセス回数を減らそうとしてVRAMに32ビットアクセスしても，少なくとも16ビットアクセス2回分程度の時間がかかってしまう，ということを意味しています．したがって，VRAMに対しては無理に32ビットアクセスをしてもあまり意味はありません．

■サンプルプログラム（ノーマルモードのみ）

このサンプルプログラムは，パレット番号9の4種類の網掛け模様で画面を埋めるプログラムです．

それらは，画面の上の方から順に，11Hと00H，11Hと44H，55Hと00H，55HとAAHをそれぞれ縦方向に交互に用いた網掛け模様です．11H=00010001B，44H=010001000B，55H=01010101B，AAH=10101010Bであることを参考にして，VRAMのデータ形式について考えてみて下さい．

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>

void wrvram(int, int);
void setpal(int, int, int, int);

void main(void)
{
    int i, x, y, addr, vdta;
    int dta[8]={0x1111, 0x0000, 0x1111, 0x4444,
                    0xaaaa, 0x0000, 0xaaaa, 0x5555};

    clrscr();
    outportb(0x6a, 1);                          /* 16色モード */
    outportb(0xa2, 0x4b);                       /* 400ラインモード */
    outportb(0xa0, 0);
    outportb(0x68, 8);
    setpal(0, 0, 0, 0);                         /* パレット番号0=黒 */
    setpal(9, 8,15, 8);                         /* パレット番号9=茶色 */
    outportb(0xa2,0xd);                         /* グラフィック画面表示開始 */
    for (i = 0; i < 4; i++) {                   /* 4種類の網掛け */
        for (y = 0; y < 100; y++) {
            for (x = 0; x < (640 / 8); x += 2) {
                addr = i * 100 * 80 + y * 80 + x;
                vdta = dta[i * 2 + (y % 2)];    /* yが偶数か奇数かで変える */
                wrvram(addr, vdta);
            }
        }
    }
}
```

```
void wrvram(int vaddr, int vdta)                    /*パレット番号9で書き込む */
{
    poke(0xa800, vaddr, vdta);
    poke(0xb000, vaddr, 0);
    poke(0xb800, vaddr, 0);
    poke(0xe000, vaddr, vdta);
}

void setpal(int pnum, int gbr, int rbr, int bbr)   /* パレット設定関数 */
{
    outportb(0xa8, pnum);
    outportb(0xaa, gbr);
    outportb(0xac, rbr);
    outportb(0xae, bbr);
}
```

■2-7-3 グラフィックのI／O

グラフィック関係のI／Oポートを表2-30に示します.

これらのI／Oポートの機能は，大きく分けて，

1） グラフィックGDCの制御
2） 表示・描画画面の設定
3） パレットの設定
4） 画面モードの設定
5） グラフィックチャージャの制御

の5つに分けられます．このうち，1）グラフィックGDCの制御と5）グラフィックチャージャの制御の具体的な方法については，関係各項を参照してください．

2）表示・描画画面の設定というのは，98の表裏2枚のグラフィック画面のうち，どちらを表示するかということと，どちらに対して書き込みを行うかを指定するものです．表示画面の指定はI／OアドレスA4Hで，描画画面の指定はA6Hで行い，それぞれについて，00Hを出力したときに表画面が，01Hを出力したときに裏画面が選択されます．たとえば，C言語で，表画面を表示しながら裏画面に書き込みを行いたいときには，

```
outportb (0xa4, 0x00) ;
outportb (0xa6, 0x01) ;
```

とすればよいのです．

3）パレットの設定というのは，前述した，各パレット番号にどの色を割り当てるか，を具体的に指定するものです．このパレットの設定方法は，カラーモードによって異なりますが，それはそれぞれ次のようになっています．

表2-30　グラフィック関係のI／Oポート（1）

リード／ライト	I／O アドレス	機　　能	D7	D6	D5	D4	D3	D2	D1	D0
リード	0A0H	GDCステータスの読み出し	←── GDCステータスフラグ ──→							
	0A2H	GDCデータの読み出し	←── GDCデータ ──→							
ライト	0A0H	GDCパラメータの書き込み	←── GDCパラメータ ──→							
	0A2H	GDCコマンドの書き込み	←── GDCコマンド ──→							
	0A4H	表示画面の指定	0	0	0	0	0	0	0	DP
	0A6H	描画画面の指定	0	0	0	0	0	0	0	WP
	0A8H	パレット番号の書き込み（16色モード）	←── パレット番号 ──→							
		パレット#3, #7の書き込み（8色モード）	#3				#7			
			0	G	R	B	0	G	R	B
	0AAH	緑輝度の書き込み（16色モード）	←── 緑輝度 ──→							
		パレット#1, #5の書き込み（8色モード）	#1				#5			
			0	G	R	B	0	G	R	B
	0ACH	赤輝度の書き込み（16色モード）	←── 赤輝度 ──→							
		パレット#2, #6の書き込み（8色モード）	#2				#6			
			0	G	R	B	0	G	R	B
	0AEH	青輝度の書き込み（16色モード）	←── 青輝度 ──→							
		パレット#0, #4の書き込み（8色モード）	#0				#4			
			0	G	R	B	0	G	R	B
	6AH	モードフリップフロップ2のコントロール（表2-31参照）	ADR6	ADR5	ADR4	ADR3	ADR2	ADR1	ADR0	DT
	7CH	GRCGモードレジスタの書き込み	←── GRCGモードレジスタ ──→							
	7EH	GRCGタイルレジスタの書き込み	←── GRCGタイルレジスタ ──→							

●16色モード時

16色モードのときには，まずI／OアドレスA8Hに色を設定したいパレット番号（0～15）を書き込み，次に，AAH，ACH，AEHにそれぞれ，そのパレット番号に割り当てたい色の緑，赤，青の輝度（0～15）を書き込みます．

たとえば，C言語で，パレット番号12（＃12と表記）の色に黄緑色（緑輝度＝15，赤輝度＝8，青輝度＝0）を割り当てたいときには，

```
outportb (0xa8, 12) ;
outportb (0xaa, 15) ;
outportb (0xac, 8) ;
outportb (0xae, 0) ;
```

とすればよいのです．

なお，PC－H98およびMATEの256色モードでは，これら4つのポートにはそれぞれ0～255までの数値を設定することができます．

●8色モード時

8色モードのときには，I／Oポートの特定の場所が特定のパレット番号に対応し，それらの部分にそれぞれの色の3原色（GRB）のありなしを設定することによって，パレットを指定します．GRBをどのようにしたときにどの色になるかは表2-25（p.83）に示してあります．具体的なパレット番号とI／Oポートの場所の対応については表2-30（p.139）を参照してください．表中で＃nと表示されているのがパレット番号nに対応する部分です．たとえば，C言語で，パレット番号＃1に青（1），パレット番号＃5（6）に黄色を割り当てたいときには，

```
outportb (0xaa, 0x16) ;
```

とすればよいのです．

4）画面モードの設定というのは，画面の解像度モードおよびカラーモードを指定するものです．画面の解像度モードの決定については，テキスト画面のところで出てきたモードフリップフロップ1と，グラフィックGDCが関係しますので，そちらを参照してください．

画面のカラーモードについては，モノクロモードかカラーモードかについてはモードフリップフロップ1が，8色モードか16色モードかについてはモードフリップフロップ2が関係します．モードフリップフロップ2については表2-31を参照してください．

たとえば，モードF／F2をコントロールして，8色グラフィックモードにしたいときには，

```
outportb (0x6a, 0x00) ;
```

とし，16色グラフィックモードにしたいときには，

```
outportb (0x6a, 0x01) ;
```

とします．

表2-31　モードフリップフロップ2に出力する値と動作の関係　（I／Oアドレス6AH）

出力する値	意　　味	解　　説
００H	８色グラフィックモード	グラフィック画面の表示モードの選択
０１H	16色グラフィックモード	
０４H	ＧＲＣＧ互換モード*	ＥＧＣの動作モードの選択
０５H	ＥＧＣ拡張モード*	
０６H	拡張モード変更不可	拡張グラフィックモード変更の可否の選択
０７H	拡張モード変更可	
４０H	ＣＲＴモード	テキスト画面の１ドットの横ずれの制御．ＣＲＴモードのときずれる
４１H	プラズマディスプレイモード	
８４H	ＧＤＣ　2.5ＭＨｚモード	グラフィックＧＤＣの動作周波数の選択．５ＭＨｚモードにするには８３Ｈと８５Ｈを両方出力する．周波数を変更したらＳＹＮＣコマンドの再設定が必要
８３Hと ８５H	ＧＤＣ　５ＭＨｚモード	

＊ 拡張モード変更可のときのみ有効

■サンプルプログラム

パレット機能を使って「緊急事態」のような効果を出します．

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>

void main(void)
{
    int br = 0, dbr = 1;

    outportb(0x6a, 1);                          /* 16色モード */
    while(kbhit() == 0) {
        br += dbr;
        if (br == 0 || br == 15) dbr = -dbr;
        while ((inportb(0xa0) & 0x20) != 0);    /* 垂直同期信号待ち */
        while ((inportb(0xa0) & 0x20) == 0);
        outportb(0xa8, 0 );                     /* パレット番号0 */
        outportb(0xaa, 0 );                     /* 緑輝度 */
        outportb(0xac, br);                     /* 赤輝度 */
        outportb(0xae, 0 );                     /* 青輝度 */
    }
}
```

グラフィックGDCは，グラフィック画面を制御しているLSIです．このGDCには，テキストGDCと同じLSIが使われており，テキストGDCとともにCRT同期信号の発生などを行うとともに，グラフィック画面の表示形式の決定や，ハードウェアによる直線・円・グラフィック文字の描画などを行います．

グラフィックGDCは，基本的にはテキストGDCと同じLSIですから，GDCへのコマンドの出力方法などについてはテキストGDCの項を参照してください．ただ，テキストGDCとグラフィックGDCでは動作モードが異なるため，若干動作の異なるコマンドやテキストGDCでは無効なので触れなかったコマンドなどがあるので，以下に，主にそのようなコマンドについての解説をします．

■動作制御系コマンド

1 RESET /グラフィックGDC動作制御

テキストGDC（p.91）と同じ．

2 SYNC /グラフィックGDC動作制御

テキストGDCのSYNCコマンド（p.91）参照．

■表示制御系コマンド

1 START /グラフィックGDC表示制御

コマンドコード

0	1	1	0	1	0	1	1	(6BH)

あるいは

0	0	0	0	1	1	0	1	(0DH)

動作　画面表示の開始を指示します．

使用例　C言語で，グラフィック画面の表示を開始するには

```
outportb (0xa2, 0x0d) ;
```

とします．

2 STOP /グラフィックGDC表示制御

コマンドコード

0	0	0	0	1	1	0	0	(0CH)

動作　画面表示の停止を指示します．

使用例　C言語でグラフィック画面の表示を停止するには，

```
outportb (0xa2, 0x0c) ;
```

とします．

3 ZOOM　/グラフィックGDC表示制御

コマンドコード

0	1	0	0	0	1	1	0

(46H)

パラメータ　P1

←ZR→	←ZW→

動作　画面表示の拡大係数と拡大描画係数を指定します．表中のパラメータの意味は以下の通りです．

◆ZR：拡大表示の倍率の指定

拡大表示の倍率－1を指定します．98では1倍(0)以外を指定すると表示が変になるので使えません．

◆ZW：拡大描画の倍率の指定

グラフィック文字描画時等の拡大描画の倍率－1を指定します．このパラメータを指定しての拡大描画は98でも正常に行うことができます．

使用例　C言語で5倍の拡大描画を指定するには，

```
outportb (0xa2, 0x46) ;
outportb (0xa0, 0x04) ;
```

とします．

4 SCROLL　/グラフィックGDC表示制御

コマンドコード

0	1	1	1	←RA→

パラメータ　RA

RA						
0	←SAD1L→					
1	←SAD1M→					
2	←SL1L→			0	0	SAD1H
3	*	IM	←SL1H→			
4	←SAD2L→					
5	←SAD2M→					
6	←SL2L→			0	0	SAD2H
7	*	IM	←SL2H→			

動作　表示画面の分割，各表示画面の表示開始アドレスの指定や表示ライン数の設定を行ないます．グラフィック画面のスムーススクロールには，通常このコマンドが用いられます．各パラメータの意味は次の通りです．

◆RA：書きかえ開始パラメータ位置の指定

このコマンドには多くのパラメータ（最大7個）があり，その都度全部のパラメータを書きかえるのではたいへんなので，このRAの部分に書きかえ始めるパラメータの位置を指定します．書きかえなかったパラメータには前の値がそのまま残ります．第1パラメータから書きかえ始めるときには，0を指定します．

◆SAD：表示開始アドレスの指定

各画面の表示開始アドレスを指定します．ここに指定するアドレスは，CPUから見たアドレスではなく，GDCから見たアドレス（テキストGDCの項参照）ですので注意してください．指定アドレスがVRAMをはみ出した場合は，ラップラウンド（再びアドレス0に戻ること）が起こります．この表示開始アドレスを1ライン分ずつずらしていくことによって，グラフィック画面をスムーススクロールさせることが可能です．

◆SL：表示ライン数の指定

各画面の表示ライン数を指定します．各画面の表示ライン数の合計がSYNCコマンドで設定されたL／Fの値（通常は400ライン）以上になるようにします．

◆ＩＭ：表示アドレスを変化させるタイミングなどの指定

ＩＭ	意　　味
0	2クロックに1回表示アドレスをインクリメントする． また，1行の表示ライン数はL／Rの設定値となる．
1	4クロックに1回表示アドレスをインクリメントする． また，1行の表示ライン数は1ラインに固定となる．

表示アドレスをインクリメントするタイミング，および1行に表示するライン数を指定します．GDC 2.5MHzのときはIM=0，5MHzのときはIM=1とします．そのために，GDC 5MHzでは1行の表示ライン数が1ラインに固定されてしまい，CSRFORMコマンドでL／Rを指定することで縦方向の倍率を変化させることはできなくなります．したがって，200ラインモードを実現するにはGDCを2.5MHzにしてIM=0とする必要があります．

◆＊：DAD+2（表示アドレスのインクリメント形態の指定）

＊	意　　味
0	“1”によるインクリメント
1	“2”によるインクリメント

98では通常，＊=0を指定します．

使用例

C言語でグラフィック画面の表示開始位置をVRAM上の200ライン目からにするには，GDCのアドレスでは1ラインが28Hなので，表示開始アドレスは28H×200＝1F40Hとなり，

```
outportb (0xa2, 0x70) ;
outportb (0xa0, 0x40) ;
outportb (0xa0, 0x1f) ;
```

とします．

5 CSRFORM　／グラフィックGDC表示制御

コマンドコード

0	1	0	0	1	0	1	1

（4BH）

パラメータ　P1

0	0	0	←――― L／R ―――→				

動作　1行のライン数の設定を行います．パラメータの意味は以下の通りです．

◆L／R：1行中のライン数の指定

1行に含まれるライン数−1を指定します．このライン数は，グラフィックGDCでは，GDCが2.5MHzのときはグラフィック画面の縦方向の拡大率に相当します．GDCが5MHzのときは意味を持ちません．ノーマルモードでは通常，400ラインモード時は0（1ライン，1倍）が，200ラインモード時は1（2ライン，2倍）が設定されています．このパラメータを操作することによって，GDCが2.5MHzならばグラフィック画面の縦方向の整数倍拡大（最大32倍）を実現することができます．

使用例　C言語でグラフィック画面表示を縦方向に4倍に拡大するには，

```
outportb (0xa2, 0x4b) ;
outportb (0xa0, 0x03) ;
```

とします（GDC 2.5MHzの場合のみ）．

6 PITCH　／グラフィックGDC表示制御

コマンドコード

0	1	0	0	0	1	1	1

（47H）

パラメータ　P1

←――― P ―――→							

動作　VRAMの横幅を，文字数（ワード数）で指定します．指定した値がそのまま文字数（ワード数）になります．ノーマルモードでは通常，GDCが5MHzのときは80（50H）が，GDCが2.5MHzのときは40（28H）が設定されています．グラフィックGDCで，Pの値としてこれらの値の倍数を指定すると，縦方向の整数分の1の縮小が実現できます．

使用例　C言語でグラフィック画面表示を縦方向に1／2に縮小するには，Pの値として28H×2＝50Hを設定すればよいので，

```
outportb (0xa2, 0x47) ;
outportb (0xa0, 0x50) ;
```

とします（GDCが2.5MHzのとき）．

1 VECTW /グラフィックGDC描画制御

コマンドコード

0	1	0	0	1	1	0	0

(4CH)

パラメータ

P1	SL	R	C	T	L	←DIR→		
P2	←DCL→							
P3	×	DGD	←DCH→					
P4	←DL→							
P5	×	×	←DH→					
P6	←D2L→							
P7	×	×	←D2H→					
P8	←D1L→							
P9	×	×	←D1H→					
P10	←DML→							
P11	×	×	←DMH→					

動作

描画する図形の種類（直線，円など）や，描画方向，描画のための各種のパラメータをセットします．各パラメータの意味は以下の通りです．

◆SL，R，C，T，L：描画図形の種類の設定

SL	R	C	T	L	描画する図形
0	0	0	0	0	1ドット
0	0	0	0	1	直線
0	0	0	1	0	傾斜しないグラフィック文字
0	0	1	0	0	円および弧
0	1	0	0	0	四辺形
1	0	0	1	0	傾斜したグラフィック文字

◆DIR：描画方向の設定

描画方向

描画方面制御パラメータ値	垂直	矩形	円弧	描画方面制御パラメータ値	垂直	矩形	円弧
0				4			
1				5			
2				6			
3				7			

注）・開始点　○終了点

描画する図形の種類によって，上の表のような意味になります．

◆DC，D，D2，D1，DM：描画情報の設定

	DC	D	D2	D1	DM
初期値	0	8	8	−1	−1
直線	｜⊿X｜	2｜⊿Y｜−｜⊿X｜	2｜⊿Y｜−2｜⊿X｜	2｜⊿Y｜	
円・弧	N	r−1	2（r−1）	−1	M
四辺形	3	A	B	−1	A

⊿X：X座標変位　⊿Y：Y座標変位　r：半径　N：描画総ドット数　M：マスキング・ドット数
A：第1辺のドット数　B：第2辺のドット数
注　Y軸方向に±45°の領域に対して直線を描画する場合には⊿Xと⊿Yの値を交換します。

描画する図形の種類によって，上の表のような意味になります．指定する値が負になる場合は，2の補数形式で指定します．

2 TEXTW /グラフィックGDC描画制御

コマンドコード

0	1	1	1	1	← RA →		

パラメータ

動作

直線, 円を描画するときの線種データ, あるいはグラフィック文字描画時の文字パターンを設定します. 1番目と2番目のパラメータは直線, 円の描画のとき線種データ, グラフィック文字描画のとき文字パターンデータとして使われます. 各パラメータの意味は次の通りです.

◆RA：書きかえ開始パラメータ位置の指定

SCROLLコマンドのRAと同様に, 書きかえる先頭のパラメータを指定します. 第1パラメータから書きかえ始めるときには, 0を指定します.

◆PTN：線種データの指定

直線・円を描くときの線のパターンを指定します. 16ドット以上の長さの線を描くときには, ここに指定したパターンが繰り返し使われます.

◆TX：文字パターンの指定

グラフィック文字を描くときの文字パターンを指定します. 8×8ドット以上の大きさの文字を描くときには, ここに指定したパターンが繰り返し使われます.

3 CSRW /グラフィックGDC描画制御

コマンドコード

0	1	0	0	1	0	0	1

(49H)

パラメータ

P1	← EADL →							
P2	← EADM →							
P3	← dAD →			0	0	EADH		

動作　描画開始点の設定を行います．各パラメータの意味は次の通りです．

◆EAD：描画開始アドレスの指定

描画開始アドレスをGDCから見たアドレス（ワード単位）で指定します．

◆dAD：ドットアドレスの指定

ドットアドレスとは，EADで指定したワードの中での，ドット単位の位置を指定するアドレスのことです．そのワードの一番左がドットアドレス0となります．

4 WRITE　／グラフィックGDC描画制御

コマンドコード

0	0	1	0	0	0	MOD

動作　ドット修正モードを指定します．ドット修正モードとは，GDCが書き込むデータと，VRAMにもとからあったデータの間で，どのような論理演算を行うかを指定するものです．パラメータの意味は次の通りです．

◆MOD：ドット修正モードの指定

MOD	意　味
0 0	REPLACE
0 1	COMPLEMENT
1 0	CLEAR
1 1	SET

REPLACEを指定すると，書きかえ対象となる領域のドットはすべてGDCのデータで置きかえられます．COMPLEMENTを指定すると，VRAMのデータとGDCのデータ間のXORが取られ，GDCのデータが1の部分の領域が反転されます．CLEARが指定されると，GDCのデータが1の部分が0になり，0の部分では元のデータがそのまま残されます．SETが指定されると，GDCのデータが1の部分が1になり，0の部分では元のデータがそのまま残されます．ごく普通の直線や円を描くときにはREPLACEを指定します．

5 VECTE　／グラフィックGDC描画制御

コマンドコード　（6CH）

0	1	1	0	1	1	0	0

動作　直線，円弧，四辺形，1ドット描画の実行開始を指示します．

6 TEXTE　／グラフィックGDC描画制御

コマンドコード　（68H）

0	1	1	0	1	0	0	0

動作　グラフィック文字描画の実行開始を指示します．

■サンプルプログラム（ノーマルモードのみ）

このサンプルプログラムは，GDCのSCROLLコマンドを使ってグラフィック画面をラップラウンドスクロール（あるライン数まで行くと1ライン目からに戻るスクロール）させるプログラムです．

グラフィック画面をラップラウンドスクロールさせるには，次の2つの方法があります．

1)PITCH幅を4000H（16384）の約数にする
2)2画面分割を使う

1)は，98のグラフィック画面がGDCアドレス4000Hでハードウェア的にラップラウンドすることを利用したものです．この方法なら，あまりテクニックを使わなくてもラップラウンドスクロールが実現できますが，通常のPITCH幅は40（80バイト=640ドット）で4000Hの約数ではありませんから，通常の画像をこの方法で正常にラップラウンドさせることはできません．

2)は，まだラップラウンドしていない領域と，すでにラップラウンドした領域を別の画面で構成することによってラップラウンドスクロールを実現するものです．たとえば，400ライン目の次に1ライン目が来るようにするには，現在の表示開始ライン位置をNとすると，

1画面目：Nライン目から表示開始，表示ライン数400-N
2画面目：1ライン目から表示開始，表示ライン数N

というように画面分割します．こうすると，1画面目で400ライン目までの画像が表示され，そのうしろに2画面目の1ライン目からの画像がくっつくので，結果的にきちんとラップラウンドができたことになります．

このプログラムでは，2)の方法を使って，現在表示されているグラフィック画像を800ライン分ラップラウンドスクロールさせます．グラフィック画面に何も表示されていないとスクロールしているかどうかわかりませんので，何かを表示してから実行するようにして下さい．

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>

void setgsta(int);

void main(void)
{
    int i;

    clrscr();
    for (i = 0; i <= 800; i += 8) setgsta(i);
}

void setgsta(int stalin)                         /*グラフィック表示位置セット関数*/
{
    int dp1sta, dp1lin, dp2sta, dp2lin, im;

    stalin %= 400;
    dp1sta = stalin * 40;
    dp1lin = (400 - stalin) << 4;
    dp2sta = 0;
    dp2lin = stalin << 4;
```

```
    if ((peek(0x0000, 0x054d) & 4) == 0) im = 0; /* GDC周波数チェック */
                                  else im = 0x40;
    while ((inportb(0xa0) & 0x20) != 0);        /* 垂直同期待ち */
    while ((inportb(0xa0) & 0x20) == 0);
    while ((inportb(0xa0) & 4) == 0);           /* FIFOの空き待ち */
    outportb(0xa2, 0x70);                       /* グラフィックGDCSCROLLコマンド */
    outportb(0xa0, dp1sta % 0x100);
    outportb(0xa0, dp1sta / 0x100);
    outportb(0xa0, dp1lin % 0x100);
    outportb(0xa0, dp1lin / 0x100 + im);
    outportb(0xa0, dp2sta % 0x100);
    outportb(0xa0, dp2sta / 0x100);
    outportb(0xa0, dp2lin % 0x100);
    outportb(0xa0, dp2lin / 0x100 + im);
}
```

■2-7-5 グラフィックBIOS

98のノーマルモードにおいて，グラフィック関係に用意されている基本入出力ルーチンには，グラフィックBIOSとグラフィックLIO（Logical Input Output）の2種類があります．グラフィックBIOSは，MS-DOSの環境でも手軽に呼び出すことができ，構造も処理も簡潔で使いやすいのですが，16色モードに対応していないのでプレーン4（インテンシティプレーン）にまったく触れられないという欠点があります．それに対し，グラフィックLIOは16色モードに対応し，BIOSよりも豊富な機能を備えていますが，もともとBASICの環境で動作することを想定したルーチンなので，MS-DOSそのほかの環境で動作させるためには面倒な手続きが必要になります．また，座標変換やクリッピング*など，BIOSよりも多くの処理を行っているため，速度面でも少し不利になります．このように，両者には一長一短あるので，場合によって使い分けるのがベストでしょう．ここでは，これらのうちのグラフィックBIOSについての解説をしていきます．グラフィックLIOのことについては「2-7-7．グラフィックLIO」の項を参照してください．

※描く図形が描画領域から外れても，描画領域内に入っている部分だけが正常に描かれるようにする処理のこと．

ノーマルモードでのグラフィック関係のBIOSは，INT 18Hによって呼び出されます．これらのBIOSコールは，画面表示やパレットレジスタなどのグラフィック関係の環境を設定するものと，グラフィックVRAMに図形そのほかを描画するものに大別されます．このうち，グラフィック関係の環境設定については，BIOSで行うよりも直接I/Oポートを制御した方が簡単なことが多いのですが，図形の描画に関してはBIOSを通した方がはるかに簡単に済みます．なぜかというと，98では普通，一般的な直線や四角形などを描くときにはグラフィックGDCを使いますが，グラフィックGDCにこのような図形を描かせるためには多数の複雑なパラメータを与えなければならないからです．そのため，一般には環境設定にはI/O直接制御，図形の描画にはBIOS（あるいはLIO）が用いられることが多いようです．そこで，以下では，環境設定のBIOSコールで，簡単にI/O制御で置きかえられるものにのみ，等価なI/O制御を併記しておくことにします．

なお，グラフィックBIOSは，以前に設定したワークエリアの値を参照するということがあまりない

ため，自前のI／O直接制御のためにグラフィックBIOSの動作がおかしくなるということは少なく，I／O直接制御とBIOSを混在させることができます．ただし，GDCを制御するBIOS（直線・円の描画など）を実行した後にグラフィックVRAMに直接アクセスするときには，GDCの描画が終わったことを確認してからアクセスを行うことが必要です．GDCとCPUが同時にVRAMにアクセスしてしまうと，VRAMにゴミが書き込まれてしまいます．具体的にGDCが描画中であるかどうかを調べる方法については，「2-6-3．テキストGDC」の項を参照してください．

さて，グラフィックBIOSの具体的な呼び出し方ですが，環境設定系のBIOSコールについては普通のBIOSコールと同じように呼び出すことができます．しかし，図形描画系のBIOSコールには，多くのパラメータや画像情報を必要とするものが多いので，そのようなBIOSコールを呼び出すときには，ユーザーはメモリ上に80バイト程度の引数・作業領域を確保し，その領域に必要な引数（パラメータ）をセットしてから呼び出しを行う必要があります．また，図形描画系のBIOSコールの中には，スタック領域を多く使うものもあるので，スタック領域は少なくとも30バイト程度は確保しておく必要があります．

■グラフィックBIOS一覧（ノーマルモード）（INT 18H）

機能コード	機　　能
４０Ｈ	グラフィック画面の表示開始
４１Ｈ	グラフィック画面の表示停止
４２Ｈ	グラフィック画面モードの設定
４３Ｈ	パレットレジスタの設定（8色パレット）
４４Ｈ	ボーダーカラーの設定
４５Ｈ	VRAMへのドット列の書き込み
４６Ｈ	VRAMからのドット列の読み出し
４７Ｈ	直線・四角形の描画
４８Ｈ	円弧の描画
４９Ｈ	グラフィック文字の描画
４ＡＨ	高速書き込みモードの設定

1　グラフィック画面の表示開始

割り込み　INT　18H

入　　力　AH←40H（機能コード）

出　　力　なし

解　　説　グラフィックGDCにグラフィック画面の表示開始を指示します．ワークエリアの書きかえなどを除けば，このコールの機能は次のようにするのと等価です．

```
outportb (0xa2, 0x0d) ;
```

2　グラフィック画面の表示停止

割り込み　INT　18H

入　　力　AH←41H（機能コード）

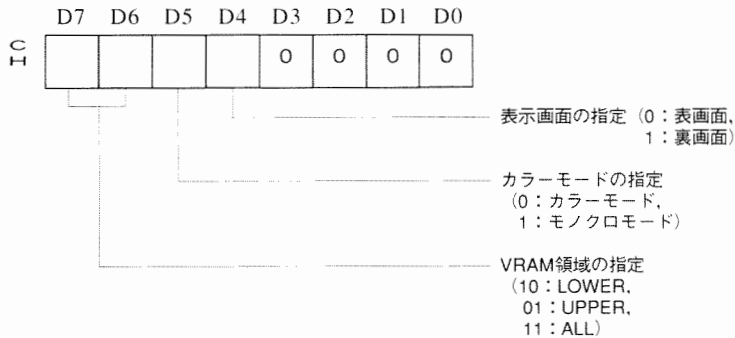
出　　力　なし

解　　説　グラフィックGDCにグラフィック画面の表示停止を指示します．ワークエリアの書きかえなどを除けば，このコールの機能は次のようにするのと等価です．

```
outportb (0xa2, 0x0c) ;
```

3　グラフィック画面モードの設定

割り込み　INT　18H

入　　力　AH←42H（機能コード）
CH←画面モード指定コード

出　　力　なし

解　　説　表示画面，カラーモード，解像度モードの設定を行います．表示画面の指定とは，表画面と裏画面のどちらをグラフィック画面に表示するかを指定するものです．0のとき表画面を，1のとき裏画面を表示します．BIOSは，I／OアドレスA4Hに値を出力することでこの設定をしています．描画画面の指定については，BIOSではサポートされておらず，LIOを使うかI／O直接制御（I／OアドレスA6H）をする必要があります．

カラーモードの指定とは，カラーモード（8色または16色モード）を使うかモノクロモードを使うかを指定するものです．0のときカラーモードが，1のときモノクロモードが指定されます．BIOSは，モードフリップフロップ1をコントロールすることでこの設

定を行っています．BIOSでは8色モードか16色モードかの設定はできません．LIOかI／O直接制御（I／Oアドレス6AH）を使います．

VRAM領域の指定とは，解像度モードとVRAMのどの部分を表示するかを指定するものです．LOWERとUPPERは解像度モードを200ラインモードに設定します．そのうえで，200ラインモードでは1枚32KBのプレーンの半分しか表示されませんから，プレーンの前半を表示するか，後半を表示するかが両者の違いです．LOWERのときプレーンの前半（表示開始アドレスはVRAMの先頭番地）を，UPPERのときプレーンの後半（表示開始アドレスはCPUから見てVRAMの先頭番地＋3E80H）を表示します．ALLのときは解像度モードが400ラインモードに設定され，プレーンの全体が表示されます．

なお，このBIOSコールは，ワークエリアの画面表示状況を参照して画面表示の開始／停止の設定をしているので，機能コード40HのBIOSコールによってグラフィック画面の表示開始が指定されていないと，そのときのハードウェアの状況に関わらずグラフィック画面の表示が停止されてしまいますので注意が必要です．

4 パレットレジスタの設定（8色パレット）

割り込み INT 18H

入　力 AH←43H（機能コード）

DS←引数・作業領域のセグメントアドレス

BX←引数・作業領域のオフセットアドレス

[BX+04H〜07H] ←パレット情報

	D7 D6 D5 D4	D3 D2 D1 D0
[BX+04H]	#6 0 G R B	#7 0 G R B
[BX+05H]	#4 0 G R B	#5 0 G R B
[BX+06H]	#2 0 G R B	#3 0 G R B
[BX+07H]	#0 0 G R B	#1 0 G R B

出　力 なし

解　説 8色モードでのパレットレジスタの設定をします．実際のパレット情報は，DS：BXで指定する引数・作業領域の，アドレス［BX+04H］〜［BX+07H］の部分に上の表のように指定します．#nと書いてあるのは，それぞれの場所に対応するパレット番号で，それぞれのパレット番号についてG（緑要素），R（赤要素），B（青要素）のありなしで割り当てる色を指定します．G，R，Bがどのような組み合わせのときにどの色が

表示されるかは表2-25（p.83）に示してあります.

また，8色・モノクロモードにおいては，これらはどのプレーンをどのように表示するかの指定になります．あるプレーンを単独表示したいとき，どの部分をどのように設定したらよいかは，次の通りです．

表示プレーン ＼ パレット番号	＃0	＃1	＃2	＃3	＃4	＃5	＃6	＃7
プレーン0	0	7	0	7	0	7	0	7
プレーン1	0	0	7	7	0	0	7	7
プレーン2	0	0	0	0	7	7	7	7

なお，このBIOSコールでは，16色モードでのパレット，16色・モノクロモードでの表示画面を設定することはできません．それらの設定をするときは，LIOか，I／O直接制御（I／OアドレスA8H～AEH）を使います．

5 ボーダーカラーの設定

割り込み INT　18H

入　力 AH←44H（機能コード）

DS←引数・作業領域のセグメントアドレス

BX←引数・作業領域のオフセットアドレス

［BX＋01H］←ボーダーカラーコード

	D7	D6	D5	D4	D3	D2	D1	D0
[BX+01H]	0	G	R	B	0	0	0	0

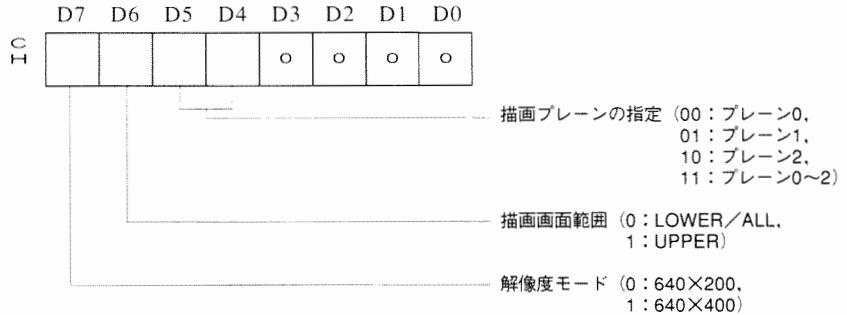
出　力 なし

解　説 CRTディスプレイで，画面表示がされる部分の周りの，表示がされない部分に付けられる色をボーダーカラーといいますが，このBIOSコールはこのボーダーカラーを設定します．ボーダーカラーは，DS：BXで指定される領域の，［BX＋1］のデータで指定します．ただし，ノーマル98では，ボーダーカラーを指定できるのは標準解像度ディスプレイ（水平解像度15KHz）を使っているときのみです．H98では，CRTのタイプによらずボーダーカラーを指定することができますが，このBIOSコールでボーダーカラーを設定することはできず，I／O直接制御（I／Oアドレス6CH）を使います．

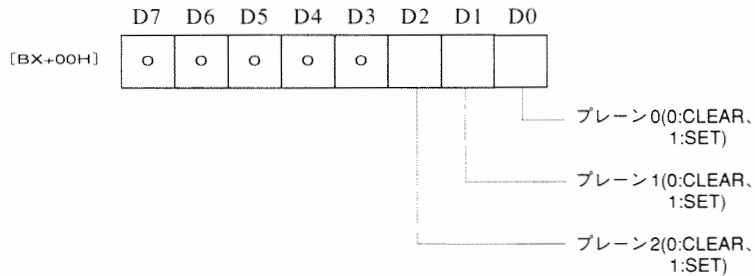
6 VRAMへのドット列の書き込み

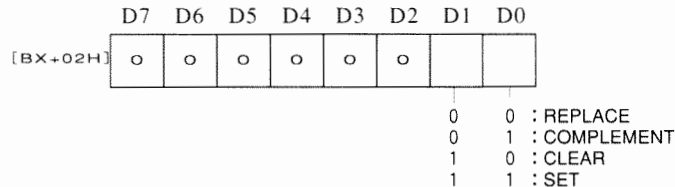
割り込み　INT　18H

入　　力　AH←45H（機能コード）

CH←対象とする描画画面の指定コード

DS←引数・作業領域のセグメントアドレス

BX←引数・作業領域のオフセットアドレス

[BX+00H] ←3プレーン同時書き込み時の描画モード

描画モードがクリアのとき，ドットパターンが1の部分は0にされ，セットのときドットパターンが1の部分は1にされる．両モードともドットパターンが0の部分には元あったデータがそのまま残される．

[BX+02H] ←単一プレーン書き込みのときの描画モード

各描画モードの意味はGDCと同じ.

[BX+08H〜09H] ←描画を開始するX座標（0〜639の範囲内）

[BX+0AH〜0BH] ←描画を開始するY座標（0〜画面下端の範囲内）

[BX+0CH〜0DH] ←描画するドット列の長さ（ドット単位）

ES←描画するドットパターンデータのセグメントアドレス

[BX+0EH〜0FH] ←描画するドットパターンデータのオフセットアドレス

[BX+2AH〜49H] ：作業領域として確保.

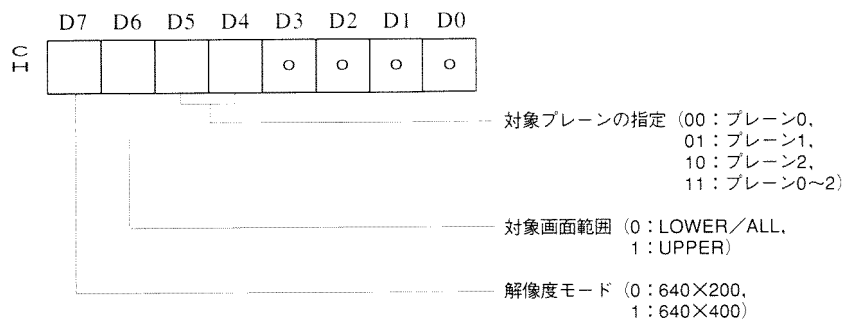
出　力　なし

解　説　グラフィックVRAMに，指定されたアドレスに格納されているドットパターンを書き込みます．ドット単位で指定された描画を開始するX，Y座標の位置から，右方向に向かって，指定されたドット数だけドットパターンを描画します．このBIOSコールは，3プレーン同時書き込みは1点の描画に，1プレーンずつの書き込みは特定領域の塗りつぶしなどによく用いられているようです．描画モードでの，REPLACE，COMPLEMENT，CLEAR，SETなどの意味は，GDCのものと同じですので，そちらを参照してください．

7　VRAMからのドット列の読み出し

割り込み　**INT　18H**

入　力　**AH←46H**（機能コード）

CH←対象とする画面の指定コード

DS←引数・作業領域のセグメントアドレス

BX←引数・作業領域のオフセットアドレス

[BX+08H〜09H] ←読み出しを開始するX座標（0〜639の範囲内）

[BX+0AH〜0BH] ←読み出しを開始するY座標（0〜画面下端の範囲内）

[BX+0CH〜0DH] ←読み出すドット列の長さ（ドット単位）

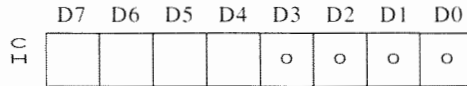
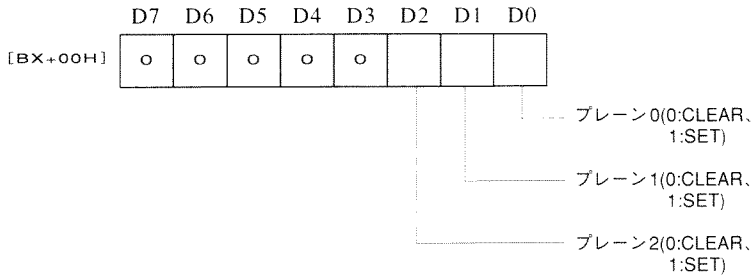
ES←読み出しデータバッファのセグメントアドレス
[BX＋10H～11H] ←読み出しデータバッファ1のオフセットアドレス
(単一プレーン，またはプレーン0用)
[BX＋12H～13H] ←読み出しデータバッファ2のオフセットアドレス
(プレーン1用，3プレーン同時読み出し時のみ)
[BX＋14H～15H] ←読み出しデータバッファ3のオフセットアドレス
(プレーン2用，3プレーン同時読み出し時のみ)
[BX＋2AH～49H] ：作業領域として確保.

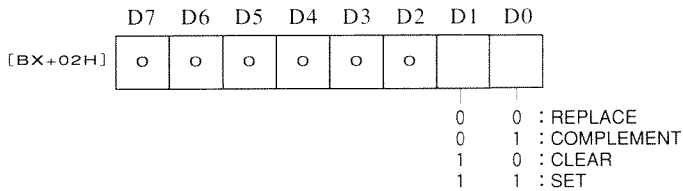
出 力 なし

解 説 VRAMの指定された位置から，指定された長さだけドット列を読み出し，読み出しデータバッファに転送します．単一プレーンの読み出しのときはデータバッファを1つだけ，3プレーン同時読み出しのときはデータバッファを1プレーンにつき1つ，合計3つ指定します．

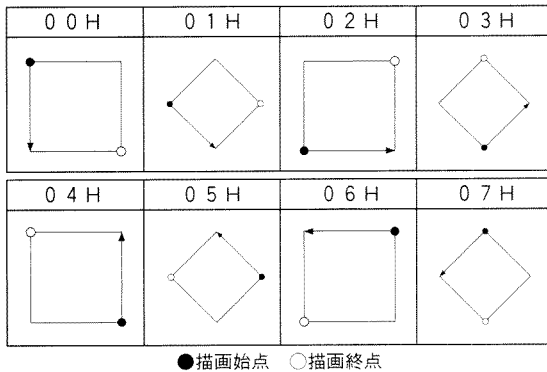
8 直線・四角形の描画

割り込み INT 18H

入 力 **AH←47H** (機能コード)
CH←対象とする描画画面の指定コード

DS←引数・作業領域のセグメントアドレス
BX←引数・作業領域のオフセットアドレス
[BX＋00H] ←3プレーン同時書き込み時の描画モード

各描画モードの意味はGDCのときと同じ.

[BX+02H] ←単一プレーン書き込みのときの描画モード

各描画モードの意味はGDCのときと同じ.

[BX+03H] ←四角形の描画の場合の描画方向

●描画始点　○描画終点

[BX+08H〜09H] ←描画開始点のX座標（0〜639の範囲内）

[BX+0AH〜0BH] ←描画開始点のY座標（0〜画面下端の範囲内）

[BX+16H〜17H] ←描画終了点のX座標（0〜639の範囲内）

[BX+18H〜19H] ←描画終了点のY座標（0〜画面下端の範囲内）

[BX+20H〜21H] ←ラインスタイル（線のドットパターン）

[BX+28H] ←描画図形の種類（1：直線,
2：四角形）

[BX+2AH〜49H] ：作業領域として確保.

出　力　なし

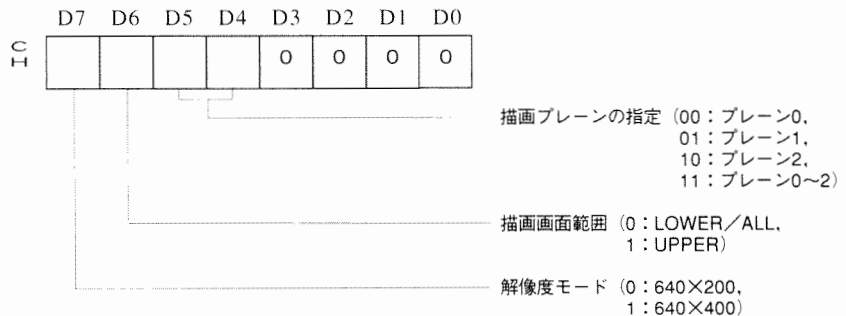
解　説　GDCにコマンドおよびパラメータを出力することによって，直線または四角形の描画を行います．直線の場合は指定した描画開始点と描画終了点だけで描かれる線は決まりますが，四角形は［BX+03H］に指定した描画方向によって描かれる図形も図のように変わってきます．

与えるパラメータのうち，ラインスタイルというのは，描画する線のドットパターンを指定する16ビットのデータです．16ビットのラインスタイルデータの各ビットの0，1が線上のドットパターンのドットのありなしに対応しており，線の長さが16ドット以上のときは，このデータが繰り返し使われます．たとえば，ここに3333Hを指定すると2ドットずつの破線が描かれ，0F0FHを指定すると4ドットずつの破線が描かれます．普通の直線にしたいときはFFFFHを指定します．

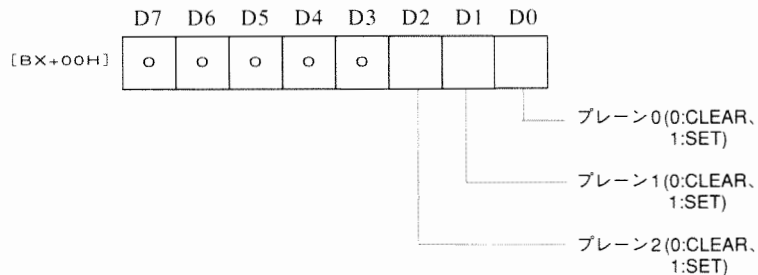
9 円弧の描画

割り込み　INT　18H

入　力　AH←48H（機能コード）

CH←対象とする描画画面の指定コード

DS←引数・作業領域のセグメントアドレス

BX←引数・作業領域のオフセットアドレス

［BX+00H］←3プレーン同時書き込み時の描画モード

各描画モードの意味はGDCのときと同じ.

[BX+02H] ←単一プレーン書き込みのときの描画モード

	D7	D6	D5	D4	D3	D2	D1	D0
[BX+02H]	0	0	0	0	0	0		

D1	D0	
0	0	: REPLACE
0	1	: COMPLEMENT
1	0	: CLEAR
1	1	: SET

各描画モードの意味はGDCのときと同じ.

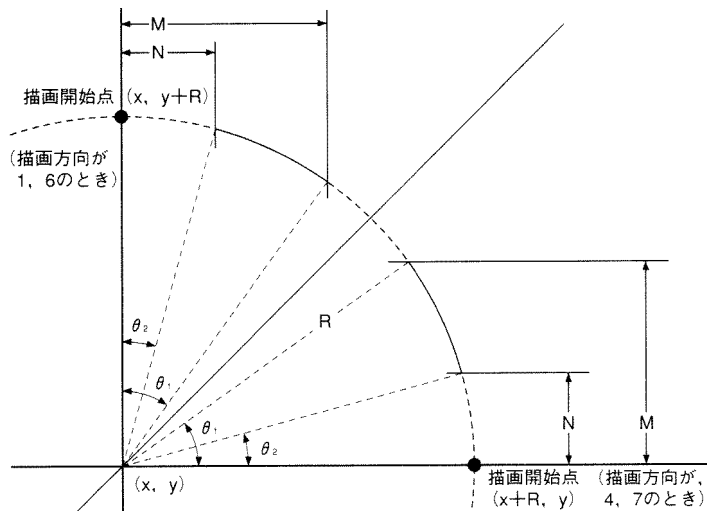
[BX+03H] ←円弧の描画方向

●描画開始点

[BX+08H～09H] ←描画開始点のX座標（0～639の範囲内）

[BX+0AH～0BH] ←描画開始点のY座標（0～画面下端の範囲内）

[BX+0CH～0DH] ←描画する総ドット数. 図のMに相当するドット数.

[BX+1AH～1BH] ←マスキングドット数. 図のNに相当するドット数.

[BX+1CH～1DH] ←円弧の半径R（ドット単位）

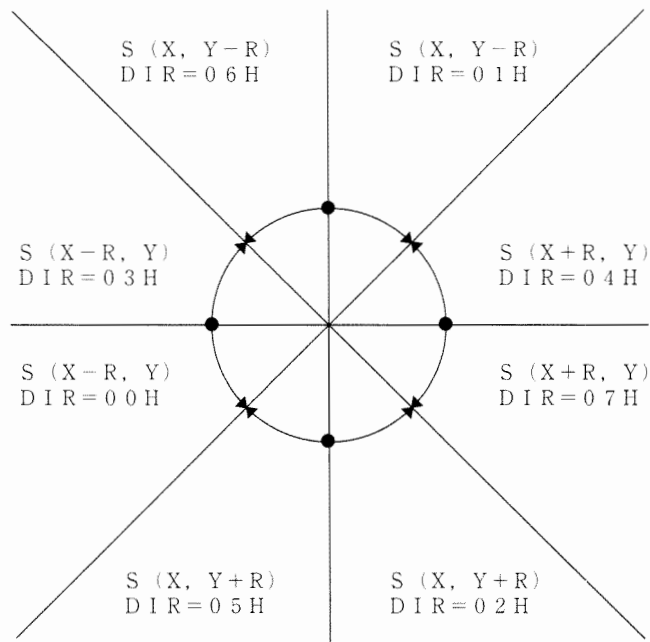
[BX+20H～21H] ←ラインスタイル（線のドットパターン）

[BX+28H] ←描画図形の種類（円弧を表す4に設定）

[BX+2AH～49H] ：作業領域として確保.

出　力　なし

解　説

GDCにコマンドおよびパラメータを出力することによって，円弧の描画を行います．1回のコールで描けるのは最大でも円の1／8なので，完全な円を描くためには，描画方向と描画開始点を変えながら8回このBIOSコールを呼び出す必要があります．1つの完全な円を描くために円のそれぞれの部分を描くとき，与えるべき描画開始点と描画方向は次のようになります．

S（○，○）　：円の中心座標を（X，Y），半径をRとしたときの描画開始点の座標

DIR　　　　　：描画方向

描画する総ドット数M，マスキングドット数Nは，描く円弧の角度位置とドット数を決定するものです．1／8円のドット数で測った全長（数学的な弧の長さとは異なる）は，$R\sin 45^\circ = R/\sqrt{2}$になりますが，Mはそのうち描画開始点から弧を描き終わるまでの区間のドット数を，Nは描画開始点から数えて実際にドットを描き始める区間までの，ドットを描かない区間のドット数を指定します．すなわち，弧を描き始める角度を$\theta 1$，弧を描き終える角度を$\theta 2$とするとM，Nは，

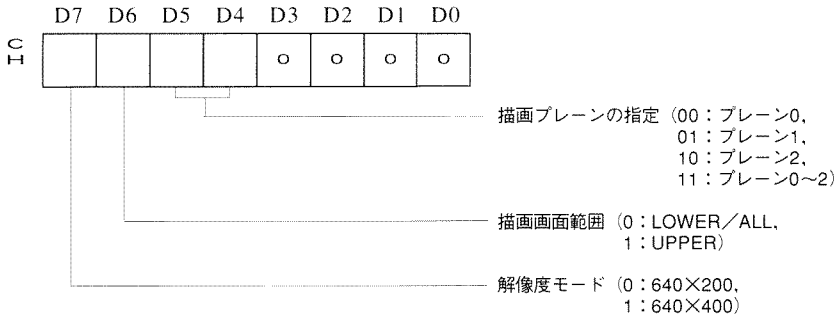
```
M=Rsinθ2
N=Rsinθ1
```

となります．弧を1／8全部描くときにはM＝R／$\sqrt{2}$≒R×0．7071，N＝0を指定します．

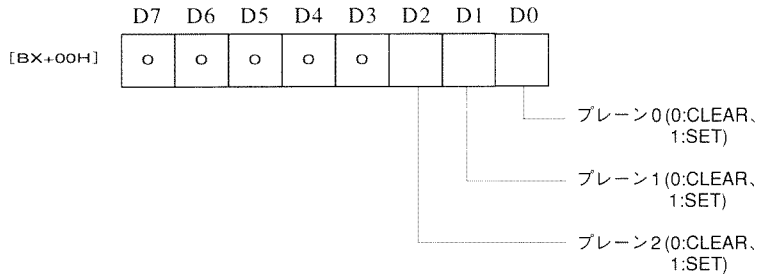
10 グラフィック文字の描画

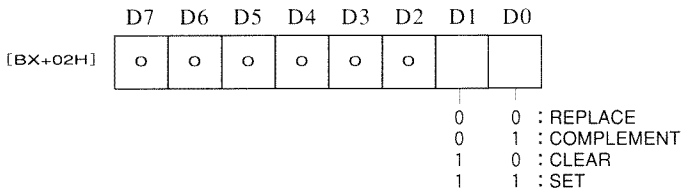
割り込み INT　18H

入　力 AH←49H（機能コード）

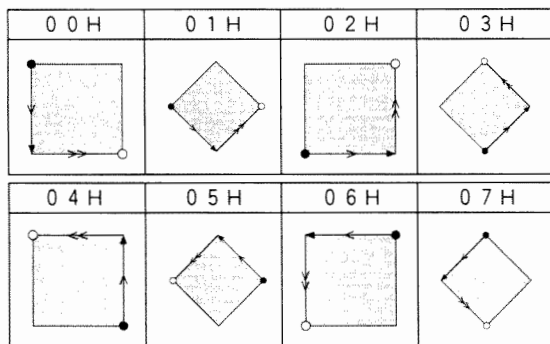
CH←対象とする描画画面の指定コード

DS←引数・作業領域のセグメントアドレス

BX←引数・作業領域のオフセットアドレス

[BX＋00H]←3プレーン同時書き込み時の描画モード

各描画モードの意味はGDCのときと同じ．

[BX＋02H]←単一プレーン書き込みのときの描画モード

各描画モードの意味はGDCのときと同じ．

[BX＋03H] ←グラフィック文字の描画方向

00H	01H	02H	03H
04H	05H	06H	07H

→ 第1描画方向　→→ 第2描画方向

[BX＋08H〜09H] ←描画開始点のX座標（0〜639の範囲内）

[BX＋0AH〜0BH] ←描画開始点のY座標（0〜画面下端の範囲内）

[BX＋0CH〜0DH] ←描画領域の第1方向ドット数．ただし描画領域が8×8ドットのときは0を指定．

[BX＋1EH〜1FH] ←描画領域の第2方向ドット数−1．ただし描画領域が8×8ドットのときは0を指定．

[BX＋20H〜27H] ←8×8ドットの文字パターンデータ

[BX＋2AH〜49H] ：作業領域として確保．

出力　なし

解説

GDCのグラフィック文字描画機能を用いて，VRAMにグラフィック文字を描画します．描画は，［BX＋20H〜27H］に与えたデータを，ごく普通（若いアドレスが上，大きい桁が左）に見たときに，そのデータの左下（［BX＋27H］の最上位ビット）から描画を始め，そこからデータの右方へ，上方へと描いていきます．したがって，そのように普通に与えたデータを普通に描画するためには，描画方向には02Hを，描画開始点には描画したい領域の左下を指定します．

指定した描画領域が8×8ドットより小さいときには，与えたデータの一部だけを使って描画が行われ，8×8ドットより大きいときには同じデータが繰り返し使われます．したがって，このBIOSコールは，特定の矩形領域の中を特定のパターンで埋める，というような目的にも使うことができます．

なお，このBIOSコールで描くグラフィック文字に関しては，GDCのZOOMコマンドの拡大描画係数が有効になります．したがって，ZOOMコマンドによって拡大係数を設定すれば，1〜16倍までの整数倍率の拡大描画が可能です．

11 高速書き込みモードの設定

割り込み INT 18H

入力 AH←4AH（機能コード）

CH←描画タイミングモード指定コード

	D7	D6	D5	D4	D3	D2	D1	D0
CH	0	0	0		0	1	1	0

D4：描画タイミングモード（0：フラッシュ描画，1：フラッシュレス描画）

出力 なし

解説

直線などの図形をGDCが描画するときに，表示期間中にも描画を行うかどうかを設定します．

フラッシュレス描画のときには，表示期間中には描画を行わず，水平および垂直同期期間中のみ描画を行います．

フラッシュ描画のときには，表示期間中にも描画を行うので，フラッシュレス描画のときの約5倍の速さで描画をすることができますが，VRAMがデュアルポートRAMでない機種では，描画中に画面がちらつきます．ちなみに，最近の機種のVRAMはすべてデュアルポートRAMなので，フラッシュ描画にしても問題は起こりません．

■サンプルプログラム（ノーマルモードのみ）

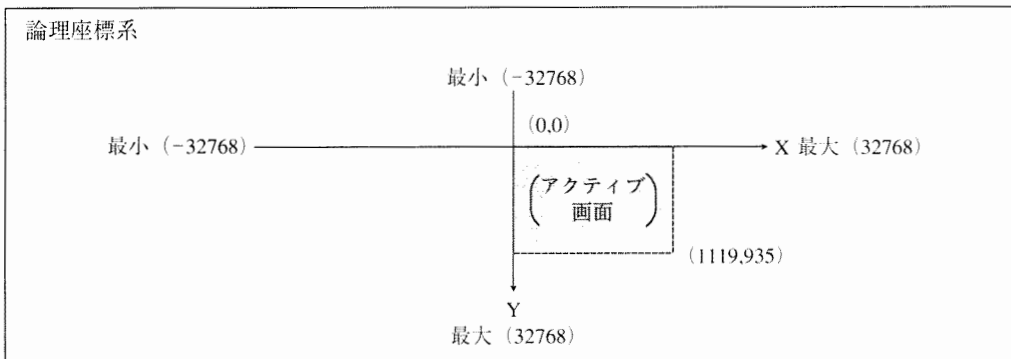
このサンプルプログラムは，グラフィックBIOSを使ってグラフィック画面表示開始・モード設定を行い，直線で図形を描くプログラムです．

このプログラムで用いたグラフィックBIOSコールはグラフィック画面の表示開始（AH=40H）とグラフィック画面モードの設定（AH=42H），それに直線・四角形の描画（AH=47H）の3つだけですが，これらの呼び出し方を参考にすれば，他のBIOSコールの呼び出し方も容易に類推できるものと思います．

```
/* ＢＩＯＳを使ってモード設定をし，直線を描く */

#include <stdio.h>
#include <conio.h>
#include <dos.h>

void line(int, int, int, int, int);
void grcolor( int );

union bufs {
   char byte;
   int word;
};

union REGS inregs, outregs;
struct SREGS segregs;
union bufs buf[40];                    /* ８０バイトのバッファ */

void main(void)
{
   int i, cl;

   clrscr();
   inregs.h.ch = 0xc0;                 /* ６４０×４００ドット */
   inregs.h.ah = 0x42;                 /* 画面モードの設定 */
   int86(0x18, &inregs, &outregs);
   outportb(0x6a, 0);                  /* ８色モード */
   outportb(0xa6, 0);                  /* 表画面に描画 */
   inregs.h.ah = 0x40;                 /* グラフィック画面表示開始 */
   int86(0x18, &inregs, &outregs);
   cl = 0;
   for (i = 0; i < 640; i += 5) {      /* 図形の描画 */
      line(0, 0, i, 399, cl % 8);
      cl++;
   }
}

void line(int x1, int y1, int x2, int y2, int cl)
{
   segregs.ds = FP_SEG(buf);
   inregs.x.bx = FP_OFF(buf);
   buf[0x00 / 2].byte = cl;
   buf[0x08 / 2].word = x1;
   buf[0x0a / 2].word = y1;
   buf[0x16 / 2].word = x2;
   buf[0x18 / 2].word = y2;
   buf[0x20 / 2].word = 0xffff;        /* ラインスタイル＝ノーマル */
   buf[0x28 / 2].byte = 1;             /* 図形＝直線 */
   inregs.h.ch = 0xb0;                 /* ３画面同時 */
   inregs.h.ah = 0x47;                 /* 直線の描画 */
   int86x(0x18, &inregs, &outregs, &segregs);
}
```

■2-7-6 ――――――――――――――――――グラフィックBIOS（ハイレゾモード）

ハイレゾモードのグラフィックBIOSは，INT 1DHによって呼び出されます．その機能は，ノーマルモードにおけるBIOSとLIOの両方の機能を含んでいますが，論理座標系（X，Y座標とも－32768～32767の範囲で指定できる座標系）を用いており，クリッピングも行っているなど，機能的にはLIOに近いものがあります．その主な特徴や実際にこれらのBIOSを呼び出すときの注意点は次のようなものです．

●ハイレゾモードの1120×936ドット（実際に1度に表示できるのは1120×750ドット）の実画面に対し，－32768～32767の整数による論理座標系を用いた座標指定を行う．

●BIOSを呼び出すときは，メモリ上に約900バイト（380Hバイト）の引数・作業領域を設定し，そこにパラメータを設定してからBIOSコールを行う．

●BIOSの使用を始める前に，BIOSコールのGINITを実行して，グラフィック環境の初期化を行う必要がある．このGINITは，引数・作業領域の初期化も行うので，それ以降のBIOSコールでは，GINITで指定して初期化した引数・作業領域を指定するようにする．

●スタックエリアは70バイト程度は必要．

●グラフィックBIOSの中には，かなり時間がかかる可能性のある処理（塗りつぶし等）があるので，そのような処理を行っている最中にも処理を中断させることができるように，BIOSは，時間のかかる処理の最中にはときどきGINITでユーザーが指定するアドレスに対してFAR CALLを行う．したがって，ユーザーは，GINITで何らかの中断処理を行うルーチン（中断処理をする必要がなければFAR RET命令だけでもよい）のアドレスを指定しておく必要がある．

なお，VRAMアクセスに対する注意はノーマルモードのときと同様です．つまり，GDCによるVRAMへの描画を行うBIOSコール（GLINE等）を行った後で，ユーザーがVRAMに直接アクセスするときには，GDCの描画が終了したことを確認してからアクセスを行う必要があります．また，ハイレゾモードのBIOSはワークエリア（作業領域）に強く依存するので，BIOSを使うときはI／O直接制御は

しない方が無難です.

■グラフィックBIOS一覧（ハイレゾモード）（INT 1DH）

機能コード	機　　能
00H	GINIT（グラフィックBIOSの初期化）
01H	GSCREEN（グラフィック画面モード設定）
02H	GVIEW（描画領域の設定）
03H	GCOLOR1（各種カラーの設定）
04H	GCOLOR2（カラーパレットの設定）
05H	GCLS（描画領域のクリア）
06H	GPSET（1点の描画）
07H	GLINE（直線・四角形の描画）
08H	GCIRCLE（円弧の描画）
09H	GPAINT1（指定色による塗りつぶし）
0AH	GPAINT2（タイルパターンによる塗りつぶし）
0BH	GGET（グラフィックパターンの読み込み）
0CH	GPUT1（グラフィックパターンの書き込み）
0DH	GPUT2（漢字パターンの書き込み）
0EH	GROLL（グラフィックパターンの移動）
0FH	GPOINT（VRAM上のドットの色の取得）
10H	GSCROLL（表示開始位置の設定）
11H	GSTART（グラフィック画面の表示開始）
12H	GSTOP（グラフィック画面の表示停止）
13H	GTERM（グラフィックBIOSの終了）

1 GINIT（グラフィックBIOSの初期化） ハイレゾ

割り込み INT 1DH

入　力 AH←00H（機能コード）

DS←引数・作業領域のセグメントアドレス

DS：［0000H～037FH］：引数・作業領域として確保

DS：［0100H～0101H］←中断処理ルーチンのオフセットアドレス

DS：［0102H～0103H］←中断処理ルーチンのセグメントアドレス

出　力 AH←結果情報（00H：正常終了,

05H：不正呼び出し）

解　説 グラフィックBIOSおよび指定された引数・作業領域を初期化し，以後，グラフィックBIOSを使えるようにします．ここでアドレスを指定した中断処理ルーチンは，塗りつぶしなどの時間がかかる処理をしているとき呼び出されます．このBIOSコールを実行すると，グラフィック環境は次のように設定されます．

・画面モードはカラーグラフィックモード.
・表示開始位置はVRAMの先頭.
・描画対象領域は全画面.
・画面表示は停止状態.

このように，GINITは画面表示を停止してしまうので，グラフィック画面を表示させるためには，GINITの後にGSCREENあるいはGSTARTを実行する必要があります.

2 GSCREEN（グラフィック画面モード設定）ハイレゾ

割り込み INT　1DH

入　力 AH←01H（機能コード）

DS←引数・作業領域（GINITで初期化済みのもの）のセグメントアドレス

DS：［0100H］←画面モード（00H：カラーグラフィックモード，
01H：モノクログラフィックモード，
FFH：現在の画面モードを変更しない）

DS：［0101H］←画面表示の有無・描画モードの設定
（00H：画面表示あり・普通描画，
01H：画面表示あり・高速描画，
02H：画面表示なし・普通描画，
03H：画面表示なし・高速描画，
FFH：現在の状態を変更しない）

DS：［0102H］←描画画面（00H：ページ0，
01H：ページ1，
02H：ページ2，
03H：ページ3，
FFH：現在の描画画面を変更しない，ただし画面モードに変更があればページ0に設定）
カラーモードのときは00Hまたは0FFHを指定.

DS：［0103H］←表示画面（00H：表示しない，
01H：画面0を表示する，
02H：画面1を表示する，
03H：画面0と1を合成して表示する，
04H：画面2を表示する，
05H：画面0と2を合成して表示する，
06H：画面1と2を合成して表示する，
07H：画面0，1，2を合成して表示する，
08H：画面3を表示する，

09H：画面0と3を合成して表示する，
0AH：画面1と3を合成して表示する，
0BH：画面0，1，3を合成して表示する，
0CH：画面2と3を合成して表示する，
0DH：画面0，2，3を合成して表示する，
0EH：画面1，2，3を合成して表示する，
0FH：画面0，1，2，3を合成して表示する，
FFH：現在の表示画面を変更しない）

出　力　AH←結果情報（00H：正常終了，
05H：不正呼び出し）

解　説　画面モードと，画面表示の有無，描画モード，表示・描画画面の設定を行います．パラメータにFFHが指定されると，その部分はGSCREENがコールされる前の状態がそのまま引き継がれます．このBIOSコールが実行されると，表示開始位置はVRAMの先頭に，描画対象領域は全画面に設定されます．

3　GVIEW（描画領域の設定）　ハイレゾ

割り込み　INT　1DH

入　力　AH←02H（機能コード）

DS←引数・作業領域（GINITで初期化済みのもの）のセグメントアドレス
DS：［0100H〜0101H］←描画領域の左上のX座標（0〜1119）
DS：［0102H〜0103H］←描画領域の左上のY座標（0〜935）
DS：［0104H〜0105H］←描画領域の右下のX座標（0〜1119）
DS：［0106H〜0107H］←描画領域の右下のY座標（0〜935）
DS：［0108H］←描画領域内を塗りつぶす色のパレット番号
（FFHを指定すると塗りつぶしを行わない）
DS：［0109H］←描画領域の外枠の色のパレット番号
（FFHを指定すると外枠を描かない）

出　力　AH←結果情報（00H：正常終了，
05H：不正呼び出し）

解　説　全表示画面領域のうちで，実際の描画の対象となる領域（描画領域，ビューポート）を指定します．ここで指定した描画領域外にはみ出た図形は，クリッピングがされて，はみ出た部分は描画されないようになります．このGVIEWによって描画領域が指定されなければ，描画領域は全画面となっています．

4 GCOLOR1（各種カラーの設定）

割り込み INT　1DH

入　力 AH←03H（機能コード）

DS←引数・作業領域（GINITで初期化済みのもの）のセグメントアドレス

DS：［0100H］←バックグラウンドカラー
（00H～0FH：カラーパレット番号指定，
FFH　　：バックグラウンドカラーを変更しない）

DS：［0101H］←フォアグラウンドカラー
（00H～0FH：カラーパレット番号指定，
FFH　　：フォアグラウンドカラーを変更しない）

出　力 AH←結果情報（00H：正常終了，
05H：不正呼び出し）

解　説 フォアグラウンドカラーとバックグラウンドカラーの設定を行います．バックグラウンドカラーというのは，画面の背景色で，BIOSコールのGCLSと，GPSETで色指定を省略したときに使われる色です．フォアグラウンドカラーというのは，そのほかのBIOSコールで色指定が省略されたときに使われる色です．

5 GCOLOR2（カラーパレットの設定）

割り込み INT　1DH

入　力 AH←04H（機能コード）

DS←引数・作業領域（GINITで初期化済みのもの）のセグメントアドレス

DS：［0100H］←色を設定するパレット番号（00H～0FH）

DS：［0101H～0102H］←設定する色情報

D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	緑輝度				赤輝度				青輝度			

あるいは

DS：［0100H］←0FFH（すべてのパレット番号に色を設定することを示す）

DS：［0101H～0120H］←設定する色情報

アドレス	内容
DS:0101H→	パレット番号0の色情報
0103H→	パレット番号1の色情報
0105H→	・ ・
011FH→	パレット番号15の色情報
0121H→	

それぞれの色情報の形式は，1つずつの設定のときと同じ．

出　力　AH←結果情報（00H：正常終了，
05H：不正呼び出し）

解　説　1つのパレット番号，あるいはすべてのパレット番号に，対応する表示色を設定します．

6　GCLS（描画領域のクリア）　ハイレゾ

割り込み　INT　1DH

入　力　AH←05H（機能コード）
DS←引数・作業領域（GINITで初期化済みのもの）のセグメントアドレス

出　力　AH←結果情報（00H：正常終了，
05H：不正呼び出し）

解　説　全描画領域を，バックグラウンドカラーで埋めます．バックグラウンドカラーが黒に設定されていれば，ごく普通の画面クリアになります．

7 GPSET（1点の描画）

割り込み INT 1DH

入　力 AH←06H（機能コード）

AL←パレット番号省略時に使う色の指定（01H：フォアグラウンドカラー，
02H：バックグラウンドカラー）

DS←引数・作業領域（GINITで初期化済みのもの）のセグメントアドレス

DS：[0100H～0101H] ←描画する点のX座標

DS：[0102H～0103H] ←描画する点のY座標

DS：[0108H] ←描画する点のパレット番号（00H～0FH）
（FFHを指定すると，ALで指定される既定色が使われる）

出　力 AH←結果情報（00H：正常終了，
05H：不正呼び出し）

解　説 VRAMの指定位置に，指定された色の1点を描画します．描画は，指定座標が描画領域の中だった場合にのみ行われます．

8 GLINE（直線・四角形の描画）

割り込み INT 1DH

入　力 AH←07H（機能コード）

DS←引数・作業領域（GINITで初期化済みのもの）のセグメントアドレス

DS：[0100H～0101H] ←描画開始点のX座標

DS：[0102H～0103H] ←描画開始点のY座標

DS：[0104H～0105H] ←描画終了点のX座標

DS：[0106H～0107H] ←描画終了点のY座標

DS：[0108H] ←描画する図形のパレット番号（00H～0FH）
（FFHを指定するとフォアグラウンドカラーで描画）

DS：[0109H] ←描画図形の種類（00H：直線，
01H：四角形，
02H：塗りつぶし四角形）

DS：[010AH] ←これ以降のパラメータ（DS：[010BH] 以降のパラメータ）の有効／無効
（00H：これ以降のパラメータはすべて無効，
01H：ラインスタイルまたは四角形塗りつぶし色が有効，
02H：タイルパターンが有効）

DS：［010BH］←　ラインスタイル下位バイト（直線または四角形のとき）
　　四角形内部を埋める色のパレット番号（塗りつぶし四角形のとき）
DS：［010CH］←ラインスタイル上位バイト（直線または四角形のときのみ有効）
DS：［010DH］←タイルパターンデータのデータ長（00H～FFH）
　　（塗りつぶし四角形のときのみ指定可能，以下のパラメータも同様）
DS：［010EH～010FH］←タイルパターンデータのオフセットアドレス
DS：［0110H～0111H］←タイルパターンデータのセグメントアドレス

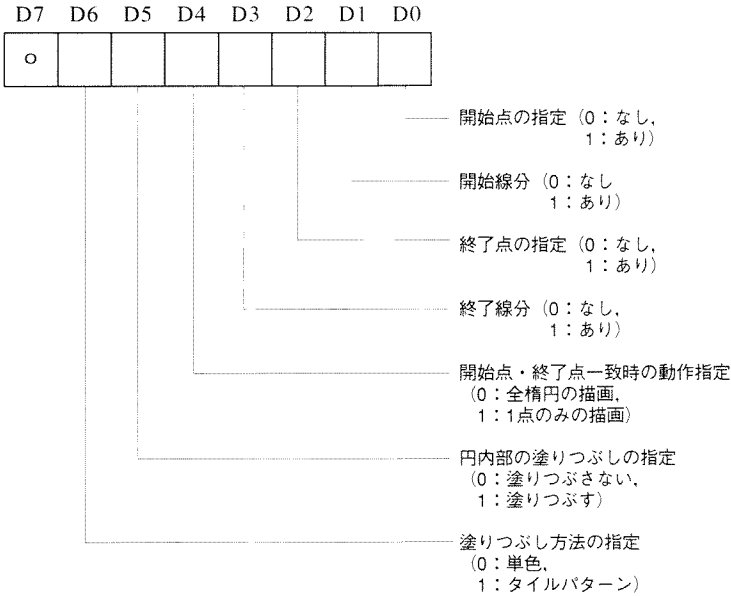
出　力　AH←結果情報（00H：正常終了，
　05H：不正呼び出し）

解　説　指定した2点間を結ぶ直線，あるいはその直線を対角線とする四角形を描画します．このBIOSコールでも，ノーマルモードの直線描画BIOSコールと同様，ラインスタイルを指定することができますが，こちらはラインスタイルの指定を省略することができます．ラインスタイルの指定を省略すると，ノーマルな直線（ラインスタイルとしてFFFFHを指定したのと同じもの）が描かれます．ラインスタイルの詳細については，ノーマルモードのBIOSコールを参照してください．

塗りつぶし四角形の描画の場合には，四角形内部の塗りつぶし方には，単一色で塗りつぶすものと，タイルパターンで塗りつぶすものが，DS：［010AH］に指定する値によって選択できます．単一色で塗りつぶす場合にはその色をDS：［010BH］に指定し，タイルパターンで塗りつぶす場合には，そのタイルパターンのデータ長とタイルパターンが格納されているメモリアドレスをDS：［010DH～0111H］に指定します．タイルパターンデータの形式については「11.GPAINT2」の項を参照してください．

9　GCIRCLE（円弧の描画）

割り込み　INT　1DH

入　力　AH←08H（機能コード）
DS←引数・作業領域（GINITで初期化済みのもの）のセグメントアドレス
DS：［0100H～0101H］←円の中心のX座標
DS：［0102H～0103H］←円の中心のY座標
DS：［0104H～0105H］←X方向の半径
DS：［0106H～0107H］←Y方向の半径
DS：［0108H］←描画する円弧のパレット番号（00H～0FH）
　　（FFHを指定するとフォアグラウンドカラーで描画）
DS：［0109H］←動作モードフラグ

DS：［010AH〜010BH］←開始点のX座標

DS：［010CH〜010DH］←開始点のY座標

DS：［010EH〜010FH］←終了点のX座標

DS：［0110H〜0111H］←終了点のY座標

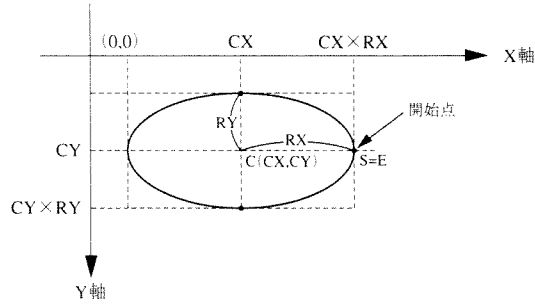
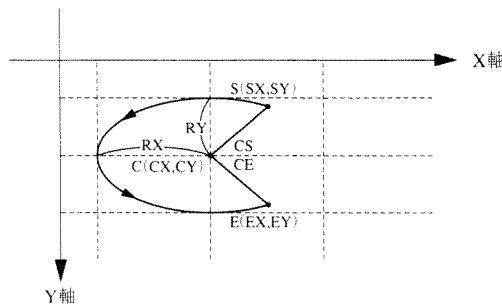
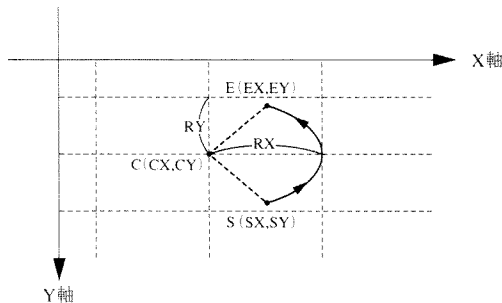
DS：［0112H］←┬ 塗りつぶす色のパレット番号（単色塗りつぶしの時）
（00H〜0FH：パレット番号，
FFH：円弧と同じパレット番号）
└ タイルパターンデータのデータ長
（タイルパターンによる塗りつぶしの時）

DS：［0113H〜0114H］←タイルパターンデータのオフセットアドレス

DS：［0115H〜0116H］←タイルパターンデータのセグメントアドレス

出　力 AH←結果情報（00H：正常終了，
05H：不正呼び出し）

解　説 円弧，楕円，扇形の描画を行います．X方向とY方向の半径に違うものを指定することで楕円を，開始点，終了点を指定することで弧を，さらに開始線分，終了線分を指定することで扇形を描画することができます．指定する各パラメータの意味は以下のようになります．

①楕円（描画開始点，終了点を指定しない場合）

②開始点S，終了点E，開始線分CS，終了線分CEの意味

③弧の描画

10 GPAINT1（指定色による塗りつぶし） ハイレゾ

割り込み **INT 1DH**

入 力 **AH←09H**（機能コード）

DS←引数・作業領域（GINITで初期化済みのもの）のセグメントアドレス

DS：［0100H～0101H］←塗りつぶし開始点のX座標

DS：［0102H～0103H］←塗りつぶし開始点のY座標

DS： [0104H] ←領域を塗りつぶす色のパレット番号（00H～0FH）
(FFHを指定するとフォアグラウンドカラーを使用)
DS： [0105H] ←塗りつぶす領域の境界色のパレット番号（00H～0FH）
(FFHを指定するとフォアグラウンドカラーを境界色と見なす)
DS： [0106H～0107H] ←塗りつぶし専用ワークエリアの終了アドレス
DS： [0108H～0109H] ←塗りつぶし専用ワークエリアの先頭アドレス
(いずれもDSをベースとしたオフセットアドレス)

出　力 **AH**←結果情報（00H：正常終了，
05H：不正呼び出し，
07H：ワークエリア不足）

解　説 指定された開始点を起点として，境界色で囲まれた領域を指定された単一色で塗りつぶします．開始点は，ビューポート内部の点でなければなりません．塗りつぶし処理には，かなり大きなワークエリアが必要なので，DS： [0106H～0107H] とDS： [0108H～0109H] でDSをベースとした専用のワークエリアを指定してやる必要があります．このワークエリアの大きさは，指定した（終了アドレス－先頭アドレス）の値で決まりますが，少なくとも16バイト以上は確保しなければなりません．塗りつぶしルーチンがこのワークエリアを使い切ってしまった場合には，その時点で処理を中断してエラーリターンしてしまうので，この専用ワークエリアは十分大きく取っておくことが必要です．

11 GPAINT2（タイルパターンによる塗りつぶし）ハイレゾ

割り込み **INT　1DH**

入　力 **AH←0AH**（機能コード）
DS←引数・作業領域（GINITで初期化済みのもの）のセグメントアドレス
DS： [0100H～0101H] ←塗りつぶし開始点のX座標
DS： [0102H～0103H] ←塗りつぶし開始点のY座標
DS： [0104H] ←タイルパターンデータのデータ長
(モノクロモードの時01H～FFH，
カラーモードの時04H～FFH)
DS： [0105H～0106H] ←タイルパターンデータのオフセットアドレス
DS： [0107H～0108H] ←タイルパターンデータのセグメントアドレス
DS： [0109H] ←塗りつぶす領域の境界色のパレット番号（00H～0FH）
(FFHを指定するとフォアグラウンドカラーを境界色と見なす)
DS： [010AH～010BH] ←塗りつぶし専用ワークエリアの終了アドレス
DS： [010CH～010DH] ←塗りつぶし専用ワークエリアの先頭アドレス
(いずれもDSをベースとしたオフセットアドレス)

出　力 AH←結果情報（00H：正常終了，
05H：不正呼び出し，
07H：ワークエリア不足）

解　説 指定された開始点を起点として，境界色で囲まれた領域を，指定されたタイルパターンで塗りつぶします．開始点は，ビューポート内部の点でなければなりません．タイルパターンデータの形式は，モノクロモードかカラーモードかによって異なり，それぞれの場合，次のようになります．

●モノクロモードの時

モノクロモードでのタイルパターンは，1バイトが横8ドットのドットパターンを表わします．その表し方は，VRAMの形式と同じです．複数バイトのタイルパターンを指定した場合には，それらがY方向に順に使われます．したがって，nバイトのタイルパターンデータが指定されると，横8ドット×縦nドットのパターンの指定になります．X方向，Y方向ともに，指定したタイルパターン以上の大きさの領域を塗りつぶすときには，同じタイルパターンが繰り返し使われます．

●カラーモードの時

カラーモードでのタイルパターンは，基本的にはモノクロモードと同じ使われ方をしますが，カラーモードではプレーンが4枚あるので，タイルパターンは4バイトが1組になっていて，それぞれの組の1バイト目がプレーン0，2バイト目がプレーン1，3バイト目がプレーン2，4バイト目がプレーン3のタイルデータとされます．したがって，カラーモードでは，モノクロモードのときの4倍のデータが必要になります．このパターンデータの組がn組指定されると，横8ドット×縦nドットのパターンの指定になります．

そのほかの，専用ワークエリアを設定するときの注意点などはGPAINT1のときと同様です．

12 GGET（グラフィックパターンの読み込み）ハイレゾ

割り込み INT　1DH

入　力 AH←0BH（機能コード）

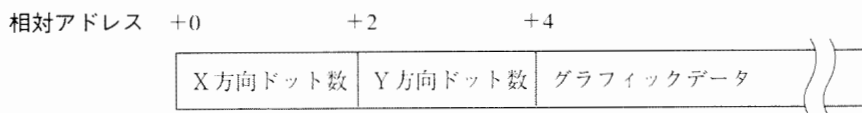
DS←引数・作業領域（GINITで初期化済みのもの）のセグメントアドレス

DS：［0100H～0101H］←読み込む領域の左上のX座標
DS：［0102H～0103H］←読み込む領域の左上のY座標
DS：［0104H～0105H］←読み込む領域の右下のX座標
DS：［0106H～0107H］←読み込む領域の右下のY座標
DS：［0108H～0109H］←データ格納バッファのオフセットアドレス
DS：［010AH～010BH］←データ格納バッファのセグメントアドレス

DS：［010CH～010DH］←データ格納バッファのサイズ（バイト単位）

出　力　AH←結果情報（00H：正常終了，
05H：不正呼び出し）

解　説　VRAMの指定領域のグラフィックデータを読み込み，指定されたデータ格納バッファに格納します．指定する領域は，ビューポートの内部でなければなりません．データ格納バッファに格納されるデータの形式は次のようなものです．

相対アドレス +0	+2	+4
X方向ドット数	Y方向ドット数	グラフィックデータ

また，データ格納バッファのサイズは，最低，次に示すバイト数必要です．式の中にでてくる領域の大きさはすべてドット単位です．

●モノクロモードの時

バッファの最低バイト数＝

（（領域のX方向の大きさ－1）￥8＋1）×（領域のY方向の大きさ）＋4

●カラーモードの時

バッファの最低バイト数＝

（（領域のX方向の大きさ－1）￥8＋1）×（領域のY方向の大きさ）×4＋4

このBIOSコールで読み込み，メモリ上に記録されたグラフィックデータは，GPUT1によって画面上の任意の位置に，読み込んだときと同じ形式で書き込むことができます．

13 GPUT1（グラフィックパターンの書き込み）ハイレゾ

割り込み　INT　1DH

入　力　AH←0CH（機能コード）

DS←引数・作業領域（GINITで初期化済みのもの）のセグメントアドレス

DS：［0100H～0101H］←書き込み領域の左上のX座標

DS：［0102H～0103H］←書き込み領域の左上のY座標

DS：［0104H～0105H］←データ格納バッファのオフセットアドレス

DS：［0106H～0107H］←データ格納バッファのセグメントアドレス

DS：［0108H～0109H］←データ格納バッファのサイズ（バイト単位）

DS：［010AH］←書き込み時に行う論理演算（00H：そのまま書き込む，
01H：NOT，
02H：AND，

03H：OR，
04H：XOR）
DS：［010BH］←グラフィックデータのモード指定
（00H：データを現在のモードのものと見なして書き込む，
01H：モノクロモードのデータをカラーモードで書き込む）
DS：［010CH］←モノクロモードのデータをカラーモードで書き込むときのフォアグラウンドカラー
DS：［010DH］←モノクロモードのデータをカラーモードで書き込むときのバックグラウンドカラー

出　力　AH←結果情報（00H：正常終了，
05H：不正呼び出し）

解　説　GGETで読み込んでおいたグラフィックデータを指定された位置に書き込みます．その際，書き込むデータと，VRAM上に元からあったデータとの間で論理演算を行うことができます．もちろん論理演算を行わず，GGETで読み込んだデータをそのまま書き込むことも可能です．

書き込むグラフィックパターンは，すべて描画領域の内部に収まっていなければなりません．書き込み領域が描画領域からはみ出ると，書き込みを行わずエラーリターンします．

パラメータ指定によって，モノクロモードで読み込んだデータをカラーモードで書き込むことが可能ですが，その際には，モノクロモードでドットがあった部分は指定されたフォアグラウンドカラーで，ドットがなかった部分は指定されたバックグラウンドカラーで書き込みが行われます．

14　GPUT2（漢字パターンの書き込み）　ハイレゾ

割り込み　INT　1DH

入　力　AH←0DH（機能コード）
DS←引数・作業領域（GINITで初期化済みのもの）のセグメントアドレス
DS：［0100H～0101H］←書き込み領域の左上のX座標
DS：［0102H～0103H］←書き込み領域の左上のY座標
DS：［0104H～0105H］←書き込む漢字のJISコード
DS：［0106H］←書き込み時に行う論理演算（00H：そのまま書き込む，
01H：NOT，
02H：AND，
03H：OR，
04H：XOR）

DS： [0107H] ←書き込みモード指定
(00H：モノクロモードで書き込む,
01H：カラーモードで書き込む)
DS： [0108H] ←カラーモードで書き込むときのフォアグラウンドカラー
DS： [0109H] ←カラーモードで書き込むときのバックグラウンドカラー

出　力 **AH**←結果情報 (00H：正常終了,
05H：不正呼び出し)

解　説 指定されたJISコードの漢字をグラフィックVRAMの指定位置に書き込みます．このとき，書き込み対象となる領域の大きさは，

全角漢字のとき横28ドット×縦30ドット
半角文字のとき横14ドット×縦30ドット

となります．書き込み対象領域は，すべて描画領域の内部に収まっていなければなりません．書き込み領域が描画領域からはみ出ると，書き込みを行わずエラーリターンします．

15 GROLL (グラフィックパターンの移動) ハイレゾ

割り込み **INT　1DH**

入　力 **AH←0EH** (機能コード)
DS←引数・作業領域 (GINITで初期化済みのもの) のセグメントアドレス
DS： [0100H～0101H] ←上下方向の移動幅 (−935～935)
DS： [0102H～0103H] ←左右方向の移動幅 (−1119～1119)
DS： [0104H] ←クリアフラグ
(00H：移動後，新しく現れた領域をパレット番号0の色でクリアする,
01H：移動後，新しく現れた領域をバックグラウンドカラーでクリアする)

出　力 **AH**←結果情報 (00H：正常終了,
05H：不正呼び出し)

解　説 VRAMに存在するグラフィックパターン全体を，指定した移動幅だけ移動します．GDCのSCROLLコマンドによる移動などとは異なり，GROLLはグラフィックパターンを実際に転送して移動させています．上下方向の移動では上方向が＋，下方向が−，左右方向では左が＋，右が−で，指定された移動幅のドット数分だけ移動が行われます．ただし，横方向の移動量は8ドット単位で，指定された移動幅が8の倍数でないときははんぱの移動幅は切り捨てられます．移動の結果，新しく現れてきた領域は，クリアフラグで指定された色で埋められます．

16 GPOINT（VRAM上のドットの色の取得）ハイレゾ

割り込み INT 1DH

入　力 AH←0FH（機能コード）

DS←引数・作業領域（GINITで初期化済みのもの）のセグメントアドレス

DS：［0100H～0101H］←色を調べるドットのX座標

DS：［0102H～0103H］←色を調べるドットのY座標

出　力 AH←結果情報（00H：正常終了，

05H：不正呼び出し）

AL←指定されたドットのパレット番号

（指定された点が描画領域を外れているとFFHが返される）

解　説 VRAM上の指定された座標のドットの色（パレット番号）を取得します．返される結果を意味のあるものにするためには，指定するドットは描画領域の内部になければなりません．

17 GSCROLL（表示開始位置の設定）ハイレゾ

割り込み INT 1DH

入　力 AH←10H（機能コード）

DS←引数・作業領域（GINITで初期化済みのもの）のセグメントアドレス

DS：［0100H～0101H］←表示開始ライン位置（0～186）

出　力 AH←結果情報（00H：正常終了，

05H：不正呼び出し）

解　説 VRAMの表示開始位置を設定します．ハイレゾモードのグラフィック画面の縦解像度は750ラインなのに対し，VRAMは936ライン分あるわけですから，差し引き186ライン分の余裕があり，その幅だけ表示領域を動かしても表示領域がVRAMからはみ出すことはないわけです．そこで，このBIOSコールは，実際に表示される750ラインの先頭ラインの位置をY座標で指定します．

18 GSTART（グラフィック画面の表示開始）ハイレゾ

割り込み INT 1DH

入力 AH←11H（機能コード）
DS←引数・作業領域（GINITで初期化済みのもの）のセグメントアドレス

出力 AH←結果情報（00H：正常終了，
05H：不正呼び出し）

解説 GDCにコマンドを送出することによって，グラフィック画面の表示を開始します．

19 GSTOP（グラフィック画面の表示停止）ハイレゾ

割り込み INT 1DH

入力 AH←12H（機能コード）
DS←引数・作業領域（GINITで初期化済みのもの）のセグメントアドレス

出力 AH←結果情報（00H：正常終了，
05H：不正呼び出し）

解説 GDCにコマンドを送出することによって，グラフィック画面の表示を停止します．

20 GTERM（グラフィックBIOSの終了）ハイレゾ

割り込み INT 1DH

入力 AH←13H（機能コード）
DS←引数・作業領域（GINITで初期化済みのもの）のセグメントアドレス

出力 AH←結果情報（00H：正常終了，
05H：不正呼び出し）

解説 グラフィックBIOSの使用を終了します．

■2-7-7 グラフィックLIO

グラフィックLIOは，グラフィックBIOSよりも高度な処理を行い，豊富な機能を備えた入出力ルーチンです．ノーマルモードのグラフィックBIOSには，16色モードに対応していないなど，機能的に十分でない部分が多くあるので，16色モードなどを用いた本格的なグラフィック処理を行おうとすると，BIOSだけでは対応できなくなってきます．そのような場合には，このグラフィックLIOを用いるのが有効になります．

LIO（Logical Input Output）というのは，BIOSのようにユーザーがハードウェアに与える値を直接指定するのではなく，ユーザーは座標などの情報を論理的な値として指定し，システムプログラム側がその値をハードウェアに与える値に変換して処理を行う，というような入出力ルーチンです．ユーザーは，LIOを使えば画面外の2点間を直線で結ぶ，といったようなことも可能で，より高度なグラフィック処理が可能です．また，たびたび述べているように，LIOはBIOSは対応していない16色モードに対応しているので，16色モードを用いたグラフィック処理を手軽に行いたいときにはLIOを使うのが便利です．

ただし，LIOにも次のような欠点があります．

1）LIOはもともとN88-BASICの環境で動作するように作られているので，MS-DOSその他の環境で動作させるときにはユーザーが割り込みベクタの設定やワークエリア（作業領域）の確保などを行う必要がある．
2）高度な処理を行っている分，処理が複雑になるので，処理速度はBIOSに比べて落ちる．

1）については，具体的な設定などの方法は後述することにします．

2）についてですが，LIOは呼び出されると，ユーザーが指定した論理的な値を実際にハードウェアに与える値に変換したうえで，多くの場合改めてBIOSを呼び出して処理を行っています．そのため，どうしても座標変換その他のよけいな処理が行われる分，処理速度は低下してしまいます．したがって，処理速度が高速であることが要求されるような場合には，機能を抑えてBIOSで済ませるか，自分でVRAMおよびGDC，グラフィックチャージャなどを制御するようにします．

さて，LIOの具体的な呼び出し方法ですが，LIOを呼び出すためには，次の4つのことが必要になります．

1）LIO用のワークエリアの確保
2）割り込みベクタテーブルのセット
3）中断処理ルーチンの用意
4）十分なスタックエリアの確保

まず1）のワークエリアの確保ですが，LIOはN88-BASICの環境で動作するときにはBASICのワークエリアを使用するので問題ないのですが，その他の環境，たとえばMS-DOSの環境で動作させるときには，ユーザーがLIOのワークエリアを確保してやる必要があります．ワークエリアの必要容量は，GCOPYコマンドを使用しないときは1200Hバイト，GCOPYコマンドを使用するときは1400Hバイトで，図2-31に示したような使われ方をします．

DS →

相対アドレス	未使用域	グラフLIO 共通作業域 128バイト	未使用域	グラフLIO 個別作業域 512バイト	未使用域	GCOPY用 作業域 128バイト	
	+0	+620H	+6A0H	+1000H	+1200H	+1380H	+1400H

注：未使用域をユーザープログラムで使用することは可能であるが，その場合には，他の領域への干渉が起こらないようにする必要がある。

図2-31 LIOのワークエリア

次に，2）の割り込みベクタテーブルのセットですが，N88-BASIC以外の環境では，LIOを呼び出すためのベクタテーブルはシステムプログラムによってセットされないので，ユーザーはLIOの使用を開始する前に，LIOを呼び出すための割り込みベクタをセットする必要があります．そのためには，LIOの各処理ルーチンのエントリアドレス（先頭番地）を知る必要がありますが，それらのエントリアドレスは，絶対メモリアドレスF9900HからのROM領域に，図2-32に示したような形式で格納されています．

そこで，ユーザーは，ベクタテーブル領域の，各コマンドを呼び出すためのベクタ番号に当たる部分（セグメントアドレス0000H：オフセットアドレスはベクタ番号×4の部分）に，ROM領域から得た各コマンド処理ルーチンのエントリアドレス（オフセットアドレス）と，各ルーチン共通のセグメントアドレス（F990H）を書き込みます．ベクタテーブルの詳細については「1-4. 割り込み」の項を参照してください．このベクタテーブルのセットは，少なくとも使用するコマンドすべてについて行う必要があります（具体例はp.201のサンプル参照）．

また，3）の中断処理ルーチンの用意というのがなぜ必要かというと，LIOのコマンドの中には塗りつぶしなどのかなり時間のかかる処理を行うものがあるので，そのような処理を実行するときには，LIOは，処理の節目節目でベクタ番号C5Hの割り込み処理ルーチンを呼び出すことによって，時間のかかる処理を途中で中断することを可能にしているのです．したがって，ユーザーは，LIOの処理中に何回か呼びだされることを前提にして，ベクタ番号C5Hの割り込み処理ルーチンを用意しておく必要がありますが，

F9900H	ベクタ番号		エントリポイントのオフセット	
+0	11（エントリ数）			
+4	A0	00	←	GINITコマンド
+8	A1	00	←	GSCREENコマンド
+12	A2	00	←	GVIEWコマンド
+16	A3	00	←	GCOLOR1コマンド
+20	A4	00	←	GCOLOR2コマンド
+24	A5	00	←	GCLSコマンド
+28	A6	00	←	GPSETコマンド
+32	A7	00	←	GLINEコマンド
+36	A8	00	←	GCIRCLEコマンド
+40	A9	00	←	GPAINT1コマンド
+44	AA	00	←	GPAINT2コマンド
+48	AB	00	←	GGETコマンド
+52	AC	00	←	GPUT1コマンド
+56	AD	00	←	GPUT2コマンド
+60	AE	00	←	GROLLコマンド
+64	AF	00	←	GPOINT2コマンド
+68	CE	00	←	GCOPYコマンド

注：各コマンドのエントリポイントのセグメントベースは，F9900H（セグメントレジスタへの格納値はF990H）

図2-32 LIOのエントリアドレス

中断処理を行う必要がなければ，これはIRET命令だけのものでも構いません．なお，この中断処理ルーチンでは一応すべてのレジスタを保証するようにするとともに，中断処理ルーチンの中でLIOを呼び出すというようなことはしないように注意してください．

最後に，4）の十分なスタックエリアの確保ですが，LIOはそれ自身もかなりのスタックエリアを必要としますが，下請けルーチンとしてBIOSを使用するのでその分のスタックエリアの消費も加わり結局130バイト程度のスタックエリアを用意しておく必要があります．

上のような準備をすれば，LIOは一応呼び出せる状態になります．そのうえで，LIOの各コマンドを使用するには，まず，GINITコマンドを実行してLIOの初期化を行います．そして，以降のコマンドを呼びだすときは，GINITコマンドで指定したDSの値と同じDSの値を指定するようにします．

なお，LIOを呼び出すと，ほとんどの場合セグメントレジスタ以外の全てのレジスタの内容が破壊されるので注意が必要です．

■グラフィックLIO一覧

割り込み	機　　能
A 0 H	GINIT（グラフィックLIOの初期化）
A 1 H	GSCREEN（グラフィック画面モード設定）
A 2 H	GVIEW（描画領域の設定）
A 3 H	GCOLOR1（各種カラーの設定）
A 4 H	GCOLOR2（カラーパレットの設定）
A 5 H	GCLS（描画領域のクリア）
A 6 H	GPSET（1点の描画）
A 7 H	GLINE（直線・四角形の描画）
A 8 H	GCIRCLE（円弧の描画）
A 9 H	GPAINT1（指定色による塗りつぶし）
A A H	GPAINT2（タイルパターンによる塗りつぶし）
A B H	GGET（グラフィックパターンの読み込み）
A C H	GPUT1（グラフィックパターンの書き込み）
A D H	GPUT2（漢字パターンの書き込み）
A E H	GROLL（グラフィックパターンの移動）
A F H	GPOINT（VRAM上のドットの色の取得）
C E H	GCOPY（グラフィックパターンの変換・格納）

1 GINIT（グラフィックLIOの初期化） LIO

割り込み INT　A0H

入　力 DS←LIOワークエリアのセグメントアドレス

出　力 AH←結果情報（00H：正常終了）

解　説 グラフィックLIOおよび指定されたワークエリアを初期化し，以後，グラフィックLIOを使えるようにします．DSに指定するワークエリアの必要容量は，GCOPYコマン

ドを使用しないときは1200Hバイト，GCOPYコマンドを使用するときは1400Hバイトで，図2-31に示したような使われ方をします．このコールを実行すると，グラフィック環境は次のように設定されます（サンプルは201ページ）．

●画面モードは8色・640×200ドットモード
●表示・描画画面とも表画面
●描画対象領域は全画面
●画面表示はあり
●パレットは標準値

2 GSCREEN（グラフィック画面モード設定） LIO

割り込み INT　A1H

入　　力 DS←LIOワークエリアのセグメントアドレス

BX←引数エリアのオフセットアドレス

DS：［BX＋00H］←画面モード

（00H：カラー・640×200ドットモード，
01H：モノクロ・640×200ドットモード，
02H：モノクロ・640×400ドットモード，
03H：カラー・640×400ドットモード，
FFH：現在の画面モードを変更しない）

DS：［BX＋01H］←画面表示の有無・描画モードの設定

（00H：画面表示あり・普通描画，
01H：画面表示あり・高速描画，
02H：画面表示なし・普通描画，
03H：画面表示なし・高速描画，
FFH：現在の状態を変更しない）

DS：［BX＋02H］←描画画面

画面モード		描画画面の指定範囲			G-VRAM使用法
		16色モード	8色モード	PC-9801U2	
カラー	640×200	0～3	0～3	0～1	G-VRAMを2つに分割
	640×400	0～1	0～1	0	G-VRAMを全て使用
モノクロ	640×200	0～15	0～11	0～5	G-VRAMを6（8）つに分割
	640×400	0～7	0～5	0～2	G-VRAMを3（4）つに分割

FFH：現在の描画画面を変更しない，ただし画面モードに変更があればページ0に設定

DS：［BX＋03H］←表示画面

表中の数字は表示する画面番号．△は全画面表示しない．＋は複数画面の合成表示．

表示画面	8色モード				16色モード			
	カラー		モノクロ		カラー		モノクロ	
	640×200	640×400	640×200	640×400	640×200	640×400	640×200	640×400
0	△	△	△	△	△	△	△	△
1	1	1	1	1	1	1	1	1
2	2		2	2	2		2	2
3			1+2	1+2			1+2	1+2
4			3	3			3	3
5			1+3	1+3			1+3	1+3
6			2+3	2+3			2+3	2+3
7			1+2+3	1+2+3			1+2+3	1+2+3
8	△	△	△	△			4	4
9			4				1+4	1+4
10			5				2+4	2+4
11			4+5				1+2+4	1+2+4
12			6				3+4	3+4
13			4+6				1+3+4	1+3+4
14			5+6				2+3+4	2+3+4
15			4+5+6				1+2+3+4	1+2+3+4
16	△	△	△	△	△	△	△	△
17	3	2	7	4			5	
18	4		8	5			6	
19			7+8	4+5			5+6	
20			9	6			7	
21			7+9	4+6			5+7	
22			8+9	5+6			6+7	
23			7+8+9	4+5+6			5+6+7	
24	△	△	△	△			8	
25			10				5+8	
26			11				6+8	
27			10+11				5+6+8	
28			12				7+8	
29			10+12				5+7+8	
30			11+12				6+7+8	
31			10+11+12				5+6+7+8	
32					△	△	△	△
33					3	2	9	5
34					4		10	16
35							9+10	5+6
36							11	7
37							9+11	5+7
38							10+11	6+7
39							9+10+11	5+6+7
40							12	8
41							9+12	5+8
42							10+12	6+8
43							9+10+12	5+6+8
44							11+12	7+8
45							9+11+12	5+7+8
46							10+11+12	6+7+8
47							9+10+12	5+6+7+8
48					△	△	△	△
49							△	
50							13	
51							14	
52							13+14	
53							15	
54							13+15	
55							13+14+15	
56							16	
57							13+16	
58							14+16	
59							13+14+16	
60							15+16	
61							13+14+15+16	
62							14+15+16	
63							13+14+15+16	

FFH：現在の表示画面を変更しない

出　力　AH←結果情報（00H：正常終了，05H：不正呼び出し）

解　説　画面モードと，画面表示の有無，描画モード，表示・描画画面の設定を行います．パラメータにFFHが指定されると，その部分はGSCREENがコールされる前の状態がそのまま引き継がれます．このBIOSコールが実行されると，描画対象領域は全画面に設定されます（サンプルは201ページ）．

3　GVIEW（描画領域の設定）LIO

割り込み　INT　A2H

入　　力　DS←LIOワークエリアのセグメントアドレス

BX←引数エリアのオフセットアドレス

DS：［BX＋00H～01H］←描画領域の左上のX座標
DS：［BX＋02H～03H］←描画領域の左上のY座標
DS：［BX＋04H～05H］←描画領域の右下のX座標
DS：［BX＋06H～07H］←描画領域の右下のY座標
DS：［BX＋08H］←描画領域内を塗りつぶす色のパレット番号
（FFHを指定すると塗りつぶしを行わない）
DS：［BX＋09H］←描画領域の外枠の色のパレット番号
（FFHを指定すると外枠を描かない）

出　　力　AH←結果情報（00H：正常終了，05H：不正呼び出し）

解　　説　全表示画面領域のうちで，実際の描画の対象となる領域（描画領域，ビューポート）を指定します．ここで指定した描画領域外にはみ出た図形は，クリッピングがされて，はみ出た部分は描画されないようになります．このGVIEWによって描画領域が指定されなければ，描画領域は全画面となっています．

4　GCOLOR1（各種カラーの設定）

割り込み　INT　A3H

入　　力　DS←LIOワークエリアのセグメントアドレス

BX←引数エリアのオフセットアドレス

DS：［BX＋01H］←バックグラウンドカラー
（00H～0FH（07H）：カラー番号指定，
FFH：バックグラウンドカラーを変更しない）
DS：［BX＋02H］←ボーダーカラー
（00H～07H：カラーパレット番号指定，
FFH：ボーダーカラーを変更しない）
DS：［BX＋03H］←フォアグラウンドカラー
（00H～0FH（07H）：カラーパレット番号指定，
FFH：フォアグラウンドカラーを変更しない）
DS:[BX+04H]←カラーモード
（00H:8色中8色モード，

01H:4096色中8色モード,
02H:4096色中16色モード)

出力 AH←結果情報(00H:正常終了,05H:不正呼び出し)

解説 フォアグラウンドカラー,ボーダーカラー,バックグラウンドカラーの設定を行います.このLIOでボーダーカラーの指定ができるのは,標準解像度(水平周波数15KHz)のディスプレイを使っているときのみです(サンプルは201ページ).

5 GCOLOR2(カラーパレットの設定) LIO

割り込み INT A4H

入力 DS←LIOワークエリアのセグメントアドレス
BX←引数エリアのオフセットアドレス

●8色中8色モードのとき
DS:[BX+00H]←色を設定するパレット番号(00H~07H)
DS:[BX+01H]←設定する色情報(00H~07H)

	D7	D6	D5	D4	D3	D2	D1	D0
DS:[BX+01H]	0	0	0	0	0	G	R	B

●4096色中8色,4096色中16色モードのとき
DS:[BX+00H]←色を設定するパレット番号(00H~0FH(07H))
DS:[BX+01H~02H]←設定する色情報

D15	D14	D13	D12	D11 D10 D9 D8	D7 D6 D5 D4	D3 D2 D1 D0
0	0	0	0	緑輝度	赤輝度	青輝度

出力 AH←結果情報(00H:正常終了,05H:不正呼び出し)

解説 指定されたパレット番号に,対応する表示色を設定します.

6 GCLS(描画領域のクリア) LIO

割り込み INT A5H

入力 DS←LIOワークエリアのセグメントアドレス

出力 AH←結果情報(00H:正常終了,05H:不正呼び出し)

解説 全描画領域を,バックグラウンドカラーで埋めます.バックグラウンドカラーが黒に設定されていれば,ごく普通の画面クリアになります(サンプルは201ページ).

7 GPSET（1点の描画） LIO

割り込み INT A6H

入　力 DS←LIOワークエリアのセグメントアドレス

BX←引数エリアのオフセットアドレス

AH←パレット番号省略時に使う色の指定（01H：フォアグラウンドカラー，
02H：バックグラウンドカラー）

DS：［BX＋00H～01H］←描画する点のX座標

DS：［BX＋02H～03H］←描画する点のY座標

DS：［BX＋04H］←描画する点のパレット番号
（FFHを指定すると，AHで指定される既定色が使われる）

出　力 AH←結果情報（00H：正常終了，05H：不正呼び出し）

解　説 VRAMの指定位置に，指定された色の1点を描画します．描画は，指定座標が描画領域の中だった場合にのみ行われます．

8 GLINE（直線・四角形の描画） LIO

割り込み INT A7H

入　力 DS←LIOワークエリアのセグメントアドレス

BX←引数エリアのオフセットアドレス

DS：［BX＋00H～01H］←描画開始点のX座標

DS：［BX＋02H～03H］←描画開始点のY座標

DS：［BX＋04H～05H］←描画終了点のX座標

DS：［BX＋06H～07H］←描画終了点のY座標

DS：［BX＋08H］←描画する図形のパレット番号（00H～0FH）
（FFHを指定するとフォアグラウンドカラーで描画）

DS：［BX＋09H］←描画図形の種類（00H：直線，
01H：四角形，
02H：塗りつぶし四角形）

DS：［BX＋0AH］←これ以降のパラメータ（DS：［BX＋0BH］以降のパラメータ）の有効／無効
（00H：これ以降のパラメータはすべて無効，
01H：ラインスタイルまたは四角形塗りつぶし色が有効，
02H：タイルパターンが有効）

DS：［BX+0BH］←┬ラインスタイル下位バイト（直線または四角形のとき）
└四角形内部を埋める色のパレット番号（塗りつぶし四角形のとき）
DS：［BX+0CH］←ラインスタイル上位バイト（直線または四角形のときのみ有効）
DS：［BX+0DH］←タイルパターンデータのデータ長（00H～FFH）
（塗りつぶし四角形のときのみ指定可能，以下のパラメータも同様）
DS：［BX+0EH～0FH］←タイルパターンデータのオフセットアドレス
DS：［BX+10H～11H］←タイルパターンデータのセグメントアドレス

出　力　AH←結果情報（00H：正常終了，05H：不正呼び出し）

解　説　指定した2点間を結ぶ直線，あるいはその直線を対角線とする四角形を描画します．このLIOでも，ラインスタイルを指定することができますが，ラインスタイルの省略も可能です．ラインスタイルを省略すると，ノーマルな直線（ラインスタイルとしてFFFFHを指定したのと同じもの）が描かれます．ラインスタイルの詳細については，ノーマルモードのBIOSコールを参照してください．

塗りつぶし四角形の描画の場合には，四角形内部の塗りつぶし方には，単一色で塗りつぶすものと，タイルパターンで塗りつぶすものが，DS：［BX+0AH］に指定する値によって選択できます．単一色で塗りつぶす場合にはその色をDS：［BX+0BH］に指定し，タイルパターンで塗りつぶす場合には，そのタイルパターンのデータ長とタイルパターンが格納されているメモリアドレスをDS：［BX+0DH～11H］に指定します．タイルパターンデータの形式については「11.GPAINT2」の項を参照してください．

なお，サンプルは201ページにあります．

9　GCIRCLE（円弧の描画）　LIO

割り込み　INT　A8H

入　力　DS←LIOワークエリアのセグメントアドレス
BX←引数エリアのオフセットアドレス
DS：［BX+00H～01H］←円の中心のX座標
DS：［BX+02H～03H］←円の中心のY座標
DS：［BX+04H～05H］←X方向の半径
DS：［BX+06H～07H］←Y方向の半径
DS：［BX+08H］←描画する円弧のパレット番号
（FFHを指定するとフォアグラウンドカラーで描画）
DS：［BX+09H］←動作モードフラグ

DS：［BX＋0AH～0BH］←開始点のX座標

DS：［BX＋0CH～0DH］←開始点のY座標

DS：［BX＋0EH～0FH］←終了点のX座標

DS：［BX＋10H～11H］←終了点のY座標

DS：［BX＋12H］←┬塗りつぶす色のパレット番号（単色塗りつぶしの時）
（FFHのとき円弧と同じパレット番号を使用）
└タイルパターンデータのデータ長（タイルパターンによる塗りつぶしの時）

DS：［BX＋13H～14H］←タイルパターンデータのオフセットアドレス

DS：［BX＋15H～16H］←タイルパターンデータのセグメントアドレス

出　力 AH←結果情報（00H：正常終了，06H：オーバーフロー）

解　説 円弧，楕円，扇形の描画を行います．X方向とY方向の半径に違うものを指定することで楕円を，開始点，終了点を指定することで弧を，さらに開始線分，終了線分を指定することで扇形を描画することができます．指定する各パラメータの意味はp.176の図を参照してください．

10 GPAINT1（指定色による塗りつぶし） LIO

割り込み INT　A9H

入　力 DS←LIOワークエリアのセグメントアドレス

BX←引数エリアのオフセットアドレス

DS：［BX＋00H～01H］←塗りつぶし開始点のX座標

DS：［BX＋02H～03H］←塗りつぶし開始点のY座標

DS：［BX＋04H］←領域を塗りつぶす色のパレット番号
（FFHを指定するとフォアグラウンドカラーを使用）

DS： [BX+05H] ←塗りつぶす領域の境界色のパレット番号

(FFHを指定すると塗りつぶしに使用する色を境界色と見なす)

DS： [BX+06H～07H] ←塗りつぶし専用ワークエリアの終了アドレス

DS： [BX+08H～09H] ←塗りつぶし専用ワークエリアの先頭アドレス

(いずれもDSをベースとしたオフセットアドレス)

出　力 **AH**←結果情報 (00H：正常終了,

05H：不正呼び出し,

07H：ワークエリア不足)

解　説 指定された開始点を起点として，境界色で囲まれた領域を指定された単一色で塗りつぶします．開始点は，ビューポート内部の点でなければなりません．塗りつぶし処理には，かなり大きなワークエリアが必要なので，DS： [BX+06H～07H] とDS： [BX+08H～09H] でDSをベースとした専用のワークエリアを指定してやる必要があります．このワークエリアの大きさは，指定した（終了アドレス－先頭アドレス）の値で決まりますが，少なくとも16バイト以上は確保しなければなりません．塗りつぶしルーチンがこのワークエリアを使い切ってしまった場合には，その時点で処理を中断してエラーリターンしてしまうので，この専用ワークエリアは十分大きく取っておくことが必要です．

11 GPAINT2（タイルパターンによる塗りつぶし） LIO

割り込み **INT　AAH**

入　力 **DS**←LIOワークエリアのセグメントアドレス

BX←引数エリアのオフセットアドレス

DS： [BX+00H～01H] ←塗りつぶし開始点のX座標

DS： [BX+02H～03H] ←塗りつぶし開始点のY座標

DS： [BX+05H] ←タイルパターンデータのデータ長

(モノクロモードの時01H～FFH,

カラーモードの時04H～FFH)

DS： [BX+06H～07H] ←タイルパターンデータのオフセットアドレス

DS： [BX+08H～09H] ←タイルパターンデータのセグメントアドレス

DS： [BX+0AH] ←塗りつぶす領域の境界色のパレット番号

(FFHを指定するとフォアグラウンドカラーを境界色と見なす)

DS： [BX+10H～11H] ←塗りつぶし専用ワークエリアの終了アドレス

DS： [BX+12H～13H] ←塗りつぶし専用ワークエリアの先頭アドレス

(いずれもDSをベースとしたオフセットアドレス)

出　力 **AH**←結果情報（00H：正常終了，
05H：不正呼び出し，
07H：ワークエリア不足）

解　説 指定された開始点を起点として，境界色で囲まれた領域を，指定されたタイルパターンで塗りつぶします．開始点は，ビューポート内部の点でなければなりません．タイルパターンデータの形式は，モノクロモードかカラーモードかによって異なり，それぞれの場合，次のようになります．

●モノクロモードの時

モノクロモードでのタイルパターンは，1バイトが横8ドットのドットパターンを表わします．その表し方は，VRAMの形式と同じです．複数バイトのタイルパターンを指定した場合には，それらがY方向に順に使われます．したがって，nバイトのタイルパターンデータが指定されると，横8ドット×縦nドットのパターンの指定になります．X方向，Y方向ともに，指定したタイルパターン以上の大きさの領域を塗りつぶすときには，同じタイルパターンが繰り返し使われます．

●カラーモードの時

カラーモードでのタイルパターンは，基本的にはモノクロモードと同じ使われ方をしますが，カラーモードではプレーンが4（3）枚あるので，タイルパターンは4バイト（3バイト）が1組になっていて，それぞれの組の1バイト目がプレーン0，2バイト目がプレーン1，3バイト目がプレーン2，4バイト目がプレーン3のタイルデータとされます．したがって，カラーモードでは，モノクロモードのときの4（3）倍のデータが必要になります（カッコ内は8色モードのとき）．このパターンデータの組がn組指定されると，横8ドット×縦nドットのパターンの指定になります．

その他の，専用ワークエリアを設定するときの注意点などはGPAINT1のときと同様です．

12 GGET（グラフィックパターンの読み込み） LIO

割り込み **INT　ABH**

入　力 **DS**←LIOワークエリアのセグメントアドレス

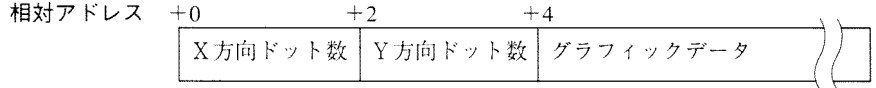
BX←引数エリアのオフセットアドレス

DS：［BX＋00H～BX＋01H］←読み込む領域の左上のX座標
DS：［BX＋02H～BX＋03H］←読み込む領域の左上のY座標
DS：［BX＋04H～BX＋05H］←読み込む領域の右下のX座標
DS：［BX＋06H～BX＋07H］←読み込む領域の右下のY座標
DS：［BX＋08H～BX＋09H］←データ格納バッファのオフセットアドレス

DS： [BX＋0AH～0BH] ←データ格納バッファのセグメントアドレス
DS： [BX＋0CH～0DH] ←データ格納バッファのサイズ（バイト単位）

出　力 **AH**←結果情報（00H：正常終了，
05H：不正呼び出し）

解　説 VRAMの指定領域のグラフィックデータを読み込み，指定されたデータ格納バッファに格納します．指定する領域は，ビューポートの内部でなければなりません．データ格納バッファに格納されるデータの形式は次のようなものです．

相対アドレス +0	+2	+4
X方向ドット数	Y方向ドット数	グラフィックデータ

また，データ格納バッファのサイズは，最低，次に示すバイト数必要です．式の中にでてくる領域の大きさはすべてドット単位です．

●モノクロモードの時
バッファの最低バイト数＝
（（領域のX方向の大きさ－1）￥8＋1）×（領域のY方向の大きさ）＋4

●8色モードの時
バッファの最低バイト数＝
（（領域のX方向の大きさ－1）￥8＋1）×（領域のY方向の大きさ）×3＋4

●16色モードの時
バッファの最低バイト数＝
（（領域のX方向の大きさ－1）￥8＋1）×（領域のY方向の大きさ）×4＋4

このBIOSコールで読み込み，メモリ上に記録されたグラフィックデータは，GPUT1によって画面上の任意の位置に，読み込んだときと同じ形式で書き込むことができます．

13 GPUT1（グラフィックパターンの書き込み）LIO

割り込み **INT　ACH**

入　力 **DS**←LIOワークエリアのセグメントアドレス
BX←引数エリアのオフセットアドレス
DS： [BX＋00H～01H] ←書き込み領域の左上のX座標
DS： [BX＋02H～03H] ←書き込み領域の左上のY座標
DS： [BX＋04H～05H] ←データ格納バッファのオフセットアドレス
DS： [BX＋06H～07H] ←データ格納バッファのセグメントアドレス

DS： [BX+08H～09H] ←データ格納バッファのサイズ（バイト単位）
DS： [BX+0AH] ←書き込み時に行う論理演算（00H：そのまま書き込む
01H：NOT,
02H：AND,
03H：OR,
04H：XOR）
DS： [BX+0BH] ←グラフィックデータのモード指定
（00H：データを現在のモードのものと見なして書き込む,
01H：モノクロモードのデータをカラーモードで書き込む）
DS： [BX+0CH] ←モノクロモードのデータをカラーモードで書き込むときの
フォアグラウンドカラー
DS： [BX+0DH] ←モノクロモードのデータをカラーモードで書き込むときの
バックグラウンドカラー

出　力 **AH**←結果情報（00H：正常終了,
05H：不正呼び出し）

解　説 GGETで読み込んでおいたグラフィックデータを指定された位置に書き込みます．その際，書き込むデータと，VRAM上に元からあったデータとの間で論理演算を行うことができます．もちろん論理演算を行わず，GGETで読み込んだデータをそのまま書き込むことも可能です．

書き込むグラフィックパターンは，すべて描画領域の内部に収まっていなければなりません．書き込み領域が描画領域からはみ出ると，書き込みを行わずエラーリターンします．

パラメータ指定によって，モノクロモードで読み込んだデータをカラーモードで書き込むことが可能ですが，その際には，モノクロモードでドットがあった部分は指定されたフォアグラウンドカラーで，ドットがなかった部分は指定されたバックグラウンドカラーで書き込みが行われます．

14 GPUT2（漢字パターンの書き込み）

割り込み **INT　ADH**

入　力 **DS**←LIOワークエリアのセグメントアドレス
BX←引数エリアのオフセットアドレス
DS： [BX+00H～01H] ←書き込み領域の左上のX座標
DS： [BX+02H～03H] ←書き込み領域の左上のY座標
DS： [BX+04H～05H] ←書き込む漢字のJISコード
DS： [BX+06H] ←書き込み時に行う論理演算（00H：そのまま書き込む,

01H：NOT,
02H：AND,
03H：OR,
04H：XOR)
DS：［BX＋07H］←書き込みモード指定
(00H：モノクロモードで書き込む,
01H：カラーモードで書き込む)
DS：［BX＋08H］←カラーモードで書き込むときのフォアグラウンドカラー
DS：［BX＋09H］←カラーモードで書き込むときのバックグラウンドカラー

出　力　AH←結果情報（00H：正常終了,
05H：不正呼び出し)

解　説　指定されたJISコードの漢字をグラフィックVRAMの指定位置に書き込みます．このとき，書き込み対象となる領域の大きさは，

全角漢字のとき横16ドット×縦16ドット
半角文字のとき横8ドット×縦16ドット
1/4角文字のとき横　8ドット×縦8ドット

となります．書き込み対象領域は，すべて描画領域の内部に収まっていなければなりません．書き込み領域が描画領域からはみ出ると，書き込みを行わずエラーリターンします．

15 GROLL（グラフィックパターンの移動）LIO

割り込み　INT　AEH

入　力　DS←LIOワークエリアのセグメントアドレス
BX←引数エリアのオフセットアドレス
DS：［BX＋00H〜01H］←上下方向の移動幅（−399〜399,
200ドットモードでは−199〜199)
DS：［BX＋02H〜03H］←左右方向の移動幅（−639〜639)
DS：［BX＋04H］←クリアフラグ（00H：移動後，新しく現れた領域をパレット番号0の色でクリアする,
01H：移動後，新しく現れた領域をバックグラウンドカラーでクリアする)

出　力　AH←結果情報（00H：正常終了,
05H：不正呼び出し)

解　説　VRAMに存在するグラフィックパターン全体を，指定した移動幅だけ移動します．GDCのSCROLLコマンドによる移動などとは異なり，GROLLはVRAM上のグラフィックパターンを実際に転送して移動させています．上下方向の移動では上方向が＋，下方向が－，左右方向では左が＋，右が－で，指定された移動幅のドット数分だけ移動が行われます．ただし，横方向の移動量は8ドット単位で，指定された移動幅が8の倍数でないときははんぱの移動幅は切り捨てられます．移動の結果，新しく現れてきた領域は，クリアフラグで指定された色で埋められます．

16 GPOINT（VRAM上のドットの色の取得）LIO

割り込み　INT　AFH

入　力　DS←LIOワークエリアのセグメントアドレス

ES←DS

BX←引数エリアのオフセットアドレス

DS：［BX＋00H～01H］←色を調べるドットのX座標

DS：［BX＋02H～03H］←色を調べるドットのY座標

出　力　AH←結果情報（00H：正常終了，

05H：不正呼び出し）

AL←指定されたドットのパレット番号

（指定された点が描画領域を外れているとFFHが返される）

解　説　VRAM上の指定された座標のドットの色（パレット番号）を取得します．返される結果を意味のあるものにするためには，指定するドットは描画領域の内部になければなりません．

17 GCOPY（グラフィックパターンの変換・格納）LIO

割り込み　INT　CEH

入　力　DS←LIOワークエリアのセグメントアドレス

AX←対象領域の左上X座標（0～639）

BX←対象領域の左上Y座標（0～399，200ラインモードのとき0～199）

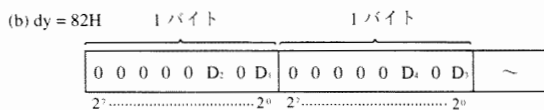
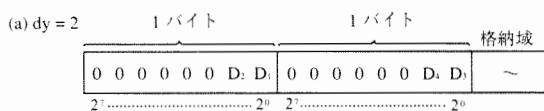
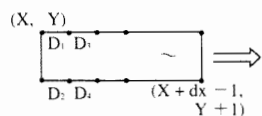
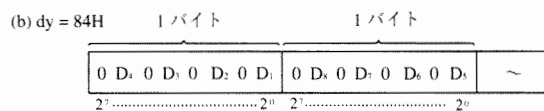
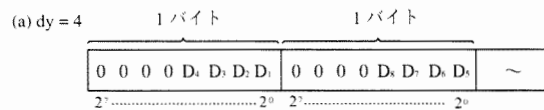
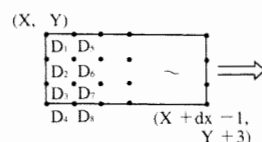
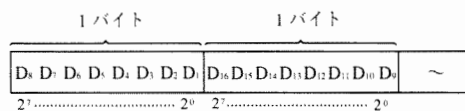
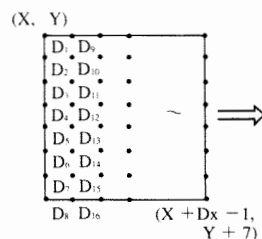
CL←対象領域のX方向ドット数（0～255）

CH←対象領域のY方向ドット数（02H，82H，04H，84H，08Hのいずれか）

DI←データ格納バッファのオフセットアドレス

ES←データ格納バッファのセグメントアドレス（DSと同じ値）

出力 **データ格納バッファ**←変換データ

解説 指定した対象領域内のグラフィックパターンを，プリンタ出力用に変換してから指定されたデータ格納バッファに格納します．98のVRAMは1バイトが横8ドットを表しますが，プリンタのイメージ形式は1バイトが縦方向の数ドット（たとえば8ドット）を表すので，このLIOはその縦横変換を行うのです．1バイトが縦の何ドットを表すようにするかはCHレジスタに指定します．データの格納形式は，CHの値によって次図のように変化します．なお，このLIOを使うときにはLIOのワークエリアは1400Hバイト確保しておくことが必要です．

③ DY = 8

■サンプルプログラム（ノーマルモードのみ）

このサンプルプログラムは，グラフィックLIOを使ってグラフィック画面のモード設定・クリアを行い，直線で図形を描くプログラムです．

このプログラムで用いたグラフィックLIOの初期化方法・コールのしかたを参考にすれば，全てのグラフィックLIOコールの使い方が類推できるものと思います．

```
/* ＬＩＯを使ってモード設定をし，直線を描く */
/* 中断処理ルーチンを用意していないので要注意 */

#include <stdio.h>
#include <conio.h>
#include <dos.h>

void line(int, int, int, int, int);

struct half {
   char low,high;
};

union bufs {
   struct half byte;
   int word;
};

union REGS inregs, outregs;
struct SREGS segregs;
union bufs buf[0x1200 / 2];                  /* １２００Ｈバイトのバッファ */

void main(void)
{
   int i, vecnum, vecoff, cl;
   char vecn;

   clrscr();
   vecn = peekb(0xf990, 0);
   for (i = 1; i <= vecn; i++) {              /* ベクタテーブルのセット */
      vecnum = peek(0xf990, i * 4    );
      vecoff = peek(0xf990, i * 4 + 2);
      poke(0x0000, vecnum * 4    ,vecoff);
      poke(0x0000, vecnum * 4 + 2,0xf990);
   }
   segregs.ds = FP_SEG(buf);
   int86x(0xa0, &inregs, &outregs, &segregs);
   inregs.x.bx = FP_OFF(buf);
   buf[0x00 / 2].byte.low  = 0x03;            /* ６４０×４００ドット */
   buf[0x01 / 2].byte.high = 0xff;
   buf[0x02 / 2].byte.low  = 0xff;
   buf[0x03 / 2].byte.high = 0xff;
   int86x(0xa1, &inregs, &outregs, &segregs);  /* ＧＳＣＲＥＥＮ */
   inregs.x.bx = FP_OFF(buf);
   buf[0x01 / 2].byte.high = 0xff;
   buf[0x02 / 2].byte.low  = 0xff;
   buf[0x03 / 2].byte.high = 0xff;
   buf[0x04 / 2].byte.low  = 0x02;            /* １６色モード */
   int86x(0xa3, &inregs, &outregs, &segregs);  /* ＧＣＯＬＯＲ１ */
   int86x(0xa5, &inregs, &outregs, &segregs);  /* ＧＣＬＳ */

   cl = 0;
   for (i = 0; i < 400; i += 5) {        /* 図形の描画 */
      line(0, 0, 639, i, cl % 16);
      cl++;
   }
}

void line(int x1, int y1, int x2, int y2, int cl)
{
   segregs.ds = FP_SEG(buf);
   inregs.x.bx = FP_OFF(buf);
   buf[0x00 / 2].word = x1;
   buf[0x02 / 2].word = y1;
   buf[0x04 / 2].word = x2;
   buf[0x06 / 2].word = y2;
   buf[0x08 / 2].byte.low  = cl;        /* パレット番号 */
   buf[0x09 / 2].byte.high = 0;         /* 図形＝直線 */
   buf[0x0a / 2].byte.low  = 0;         /* 以降のパラメータ無効 */
   int86x(0xa7, &inregs, &outregs, &segregs);
}
```

§2-8 メモリ

98でMS-DOSを利用する場合，通常のメモリ空間は1Mバイトまでしか使えません．8086やV30といったCPUでは，もともとCPUが管理できるメモリ容量が1Mバイトしかありませんから，これは当然のこととも言えるのですが，80286以上のCPUでは1Mバイトを越えるアドレス空間を持っているのに（80286，386SXは16Mバイト，386DX以上では，最大4Gバイト）*，MS-DOSを使っている限り，1Mバイトが上限となってしまうのです．これはMS-DOSというOSが1Mバイト以上のメモリの管理を想定していなかったためです．

(*)ただし，98では14.6Mバイトを越える増設は不可能になっている場合がほとんどです．

この1Mバイトの空間には，一般に「メインメモリ」と呼ばれる640KバイトのRAMや，テキストVRAM，グラフィックVRAM，BIOS-ROMなどが割り当てられています．通常，MS-DOS上のプログラムの動作は，すべてこの空間で行われます．

しかし，最近のアプリケーションソフトはどんどん大型化しており，この1Mバイトという空間が非常に狭く感じられるようになってきました．そこで，MS-DOS上でも何とか1Mバイト以上のメモリを使えないものか，と考えられたのがEMS，XMSなどの拡張メモリの管理方式です．

これらの拡張メモリ管理方式の基本原理はよく似ていていて，要するに，MS-DOSで通常アクセスできる1Mバイトのメモリ空間のどこかに「パイプ」あるいは「窓」のようなもの作り，そのパイプや窓を通して，広大な拡張メモリ空間にアクセスをするというものです．ただし，EMSとXMSでは，そのパイプ・窓の開けかたが異なります．

EMSは先ほど説明した1Mバイトのメモリ空間の中の拡張ROM領域に窓を設定し，CPUとは独立した拡張メモリを利用できるようにします．この方式ですと，要するに1Mバイトのメモリ空間にアクセスできればよいのですから，CPUが80486であろうと8086であろうと同様に利用できます．

これに対してXMSは，1Mバイトのメモリ空間から1Mバイト以上のメモリ空間（これを「プロテクトメモリ」と呼びます）を間接的に制御する方法です．本来，この領域のメモリにアクセスするためには，CPUをプロテクトモード（エンハンストモード）にする必要があり，MS-DOSからこの領域のメモリは直接は利用できません**．そこで，専用のドライバソフトを使用し，これを通してプロテクトメモリをアクセスするのがXMSです．なお，MS-DOS ver.5からは，DOS本体をこの領域の一部分に置くことが可能になっています．

(**)MS-DOSはリアルモードで動いています．

このXMSはプロテクトメモリを使用するものなので，1Mバイト以上のメモリ空間を持つCPU（80286以上）でなければ使用できません．つまり，8086やV30では使えないわけです．XMSはEMSと

違い，連続なメモリ領域を取ることが可能であるという利点がありますが，旧型のマシンでは使えないのが欠点でしょう．

また，CPUが80386以上のマシンでは，プロテクトメモリを仮想的に，前述の1Mバイトの空間に配置することが可能になっています（仮想86モード）．これにより，386以上のマシンでは，プロテクトメモリをEMSとして利用するといったことが可能になります．

ここでは，MS-DOSにおいて，基本的な，メインメモリや，EMSメモリの利用法などを説明します．

■2-8-1 メインメモリ

メインメモリは，コンベンショナルメモリとも呼ばれ，MS-DOSのプログラムが，実際に，ロードされ実行される領域です．MS-DOS本体や，デバイスドライバなども，この領域にロードされ，動作します．なお，DOSやドライバは，ほかのメモリ領域に置くことも可能となっていますが，このことについては，後述します．

また，アプリケーションにより，この領域のメモリを確保し，利用することも可能です．DOSでは一番扱いやすいメモリ領域です．メインメモリをアプリケーションから確保するためのファンクションコールを以下に示します．これは，DOSファンクションコールに含まれます．

■メインメモリ関係のDOSファンクションコール（INT 21H）

機能コード	機能	ノーマル	ハイレゾ
4 8 H	メモリの割り当て	○	○
4 9 H	割り当てられたメモリの開放	○	○
4 A H	割り当てられたメモリブロックの変更	○	○

詳しい利用方法は，「4-7　MS-DOSファンクションコール一覧」を参照してください．

■2-8-2 EMS

EMSメモリは，データの記憶に多くのメモリを必要とする場合に，非常に有用で多くのアプリケーションで利用されています．また，データの格納に限らず，このメモリ上でのプログラムの実行も可能です．ここでは，EMSのファンクションおよびその利用法を説明します．なお，EMSは，EMM386.EXEなどのEMSドライバを組み込まないと利用できません．

EMSメモリは，ある連続した64Kバイトの空間に割り付けられます．これを4つに区切り，16Kバイト単位で，メモリを入れ替えることにより，多くのメモリが利用できる仕組みになっています．98では一般に，EMSのための連続した，64Kバイトのフレームは，C000Hセグメントに割り当てられます．EMSは図2-33に示したようにして多くのメモリをアクセスします．ただし，これからわかるように，1度にアクセスできるメモリは，64Kバイトが最大となります．

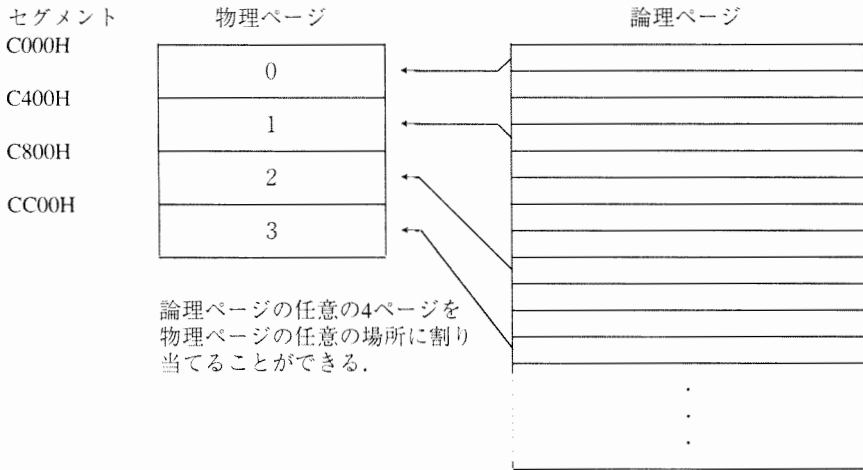



図2-33　EMSの構造

EMSには，286以下のマシンで使う，ハードウェアEMSとよばれる，ハードウエアでEMSの機能を実現するものや，386以上のマシンで仮想86モードを利用して，プロテクトメモリをEMSとして見せかける方法などがありますが，どちらの場合も，EMSを利用する場合のファンクションは同じで，区別なく利用することができます．

●EMSファンクションの利用方法

EMSファンクションの利用の手順は次のようになります．

①EMMドライバがインストールされているかチェックする
②メモリの割り当てを行う．
③EMMハンドルを取得する．
④マッピングを行う．
⑤実際のメモリの読み書きを行う．

もちろん，②以降を実行するためには，EMMドライバが組み込まれていなければなりません．

①のEMMドライバがインストールされているかチェックする方法には，オープンハンドル法と，ゲットインタラプトベクタ法の2種類があります．

オープンハンドル法は，DOSのファンクションコール3DHを用いて，EMMXXXX0というデバイスのオープンを行います．オープンに成功したら，これがドライバかファイルかを判断するために，DOSのファンクションコール44Hを用います．これによりデバイスと判断されると，EMMドライバがインストールされているということになります．

ゲットインタラプトベクタ法は，DOSのファンクションコール35Hを用い，INT67Hの割り込みベクタ（セグメントのみでよい）を取得します．このセグメントにおける，オフセット0AHから始まる8バイトの文字列がEMMXXXX0であることをチェックします．この文字列が存在すれば，EMMドライバ

がインストールされているということになります．

一般に，どちらのチェック法を使っても構わないのですが，デバイスドライバからEMMファンクションを使うときなどは，ゲットインタラプトベクタ法でないといけません．最後に示すサンプルプログラムでは，この方法でEMMファンクションのインストール状況を調べています．具体的な調べる方法は，サンプルプログラムを参照してください．

②のメモリの割り当てというのは，そのプログラムで使用するEMSの容量を指定するもので，EMSファンクションの機能コード43Hで実現します．確保するメモリの単位は「ページ」（1ページ=16Kバイト）です．このメモリの割り当てを実行すると，次に説明するEMMハンドルを取得できます．

③のEMMハンドルというのは，EMS使うプログラムの認識番号のようなものです．通常，EMSメモリは複数のプログラムで使用されることが多く，たとえば，4MバイトのEMSの内，日本語FEPに100Kバイト，ディスクキャッシュに1Mバイト，常駐ソフトに数Kバイト，残り2Mバイト弱をアプリケーションソフトに使用する，などという使われ方をします．したがって，EMSを操作するとき，どのプログラムがEMSを操作しようとしているのか，EMMドライバに知らせてやらなければなりません．これを間違えますと，FEPのデータがワープロソフトの中にあらわれてしまう，といったようなおかしなことになりかねません．この識別のための番号がEMMハンドルで，普通は組み込まれた順番に，1，2，3，…と番号が振られていきます．

このEMMハンドルは，②のメモリの割り当てを実行すると取得できます．

④のマッピングというのは，EMSの物理ページと論理ページの対応をつけるものです．図2-33にあるように，物理ページは4ページ（64Kバイト）しかありませんから，このままで，数MバイトにもおよぶEMS全体を操作することはできません．そこで，物理ページに論理ページを割り付け，必要に応じてこの割り付けを変更します．

たとえば，物理ページ0～3に論理ページ0～3を割り付けます．この状態でEMSメモリにデータを保存した後，今度は物理ページ0～3を論理ページ4～7に割り付けます．そしてこの状態でもEMSメモリにデータを保存します．そして必要に応じて論理ページ0～3と4～7を切り替えます．こうすることによって，0～7ページ（128Kバイト）のデータが使用できることになります．

なお，このページの割り付けは原理的には1ページごとに行えますが，実際には4ページ単位で行われることがほとんどです．

⑤の実際のメモリの読み書きは，それぞれのプログラミング言語によって実現のしかたが異なります．Cでは，物理ページの存在するアドレスに配列を割り付けたり，ポインタを利用するのが一般的な方法でしょう．詳しくはサンプルプログラムを参考にしてください．

■EMSファンクション一覧（INT 67H）

機能コード	機能
4 0 H	マネージャのステータス照会
4 1 H	ページフレームセグメントの取得
4 2 H	未アロケートページカウントの取得
4 3 H	ページの割り当て
4 4 H	ハンドルページのマップ，アンマップ
4 5 H	ページの開放
4 6 H	バージョンの取得
4 7 H	ページマップのセーブ
4 8 H	ページマップの復帰
4 B H	ハンドル数の取得
4 C H	ハンドル・ページの取得
4 D H	全ハンドル・ページの取得
4 E H	ページマップの取得 ページマップの設定 ページマップの取得と設定 ページマップ格納配列のサイズ取得
4 F H	一部のマッピング情報の取得 一部のマッピング情報の復帰 一部のマッピング情報を格納する配列のサイズ取得
5 0 H	複数頁のマップ，アンマップ
5 1 H	ページの再割当
5 2 H	ハンドル属性の取得 ハンドル属性の設定 不揮発性属性のサーポート可能性の調査
5 3 H	ハンドル名の取得 ハンドル名の設定
5 4 H	ハンドルのディレクトリ情報の取得 指定の名前を持つハンドルの検索 ハンドル総数の取得
5 5 H	ページマップの変更とジャンプ
5 6 H	ページマップの変更とコール ページマップの変更に必要なスタックサイズの取得
5 7 H	メモリ領域のコピー メモリ領域の交換
5 8 H	マップ可能な物理アドレス配列の取得 マップ可能な物理アドレス配列のエントリ数の取得

5 9 H	ハードウェア構成に関する情報の取得 未アロケートのＲＯＷページ数の取得
5 A H	標準サイズのページの割り当て ＲＯＷページの割り当て
5 B H	代替マップレジスタセットの取得 代替マップレジスタセットの設定 代替マップセーブ配列のサイズの取得 代替マップレジスタセットの割り当て 代替マップレジスタセットの開放 ＤＭＡレジスタセットの割り当て 代替マップレジスタによるＤＭＡの使用許可 代替マップレジスタに対応するＤＭＡの使用禁止 ＤＭＡレジスタセットの開放
5 C H	ウォームブート用の拡張メモリハードウェア準備
5 D H	OS/Eファンクションセットの使用許可の設定 OS/Eファンクションセットの使用禁止の設定 アクセスキーのリターン
7 0 H	ページフレーム用バンクのステータス取得 ページフレーム用バンクの状態の設定

各出力のステータスコードは，「表2-32　ステータスコード一覧」を参照してください．

表2-32　ステータスコード一覧

ステータスコード	機能
0 0 H	正常終了
8 0 H	拡張メモリ管理プログラムが動作せず
8 1 H	拡張メモリハードウェアが動作せず
8 3 H	指定のＥＭＭハンドルが見つからない
8 4 H	ファンクションコードが未定義
8 5 H	すべてのＥＭＭハンドルが使用中
8 6 H	マッピングコンテキストの復元エラー
8 7 H	要求された量のページが存在しない
8 8 H	要求された未アロケートページが存在しない
8 9 H	０ページはハンドルに割り当てられない
8 A H	マップする論理ページが範囲外
8 B H	物理ページが範囲外
8 C H	マップレジスタコンテキスト保存領域がいっぱい
8 D H	スタックにハンドルに関連したコンテキストが存在

8EH	スタックに指定されたハンドルに関連したコンテキストが存在しないEMBの再割り当て
8FH	サブファンクションパラメータが未定義
90H	属性の型が未定義
91H	不揮発性をサポートしていない
92H	拡張メモリのコピー先と元が同じハンドルで重複
93H	拡張メモリのコピー先と元の大きさが異なる
94H	標準メモリと拡張メモリの領域が重複
95H	論理ページ内のオフセットが論理ページの大きさよりより大きい
96H	領域の大きさが1Mバイトを超えた
97H	拡張メモリの交換元と先が同じハンドルで重複
98H	メモリタイプが未定義
9AH	指定した副ページマップレジスタが未サポート
9BH	すべての副ページマップレジスタがアロケート済み
9CH	副ページマップレジスタセットが未サポート
9DH	指定された副ページマップレジスタセットが未定義か未アロケート
9EH	専用のDMAチャネルが未サポート
9FH	指定したDMAチャネルが未サポート
A0H	指定したハンドル名に対するハンドル値がない
A1H	指定したハンドル名がすでに存在する
A2H	移動，交換のときに1Mバイトのアドレス空間を超えようとした
A3H	ファンクションに渡したデータ構造が不正
A4H	OSが指定のファンクションのアクセスを拒否

1 マネージャのステータス照会 EMS

割り込み INT 67H

入　力 AH←40H

出　力 AH→ステータスコード（表3-32：p.207参照）

解　説 メモリマネージャの存在などをチェックします．

サンプル EMMマネージャのステータスを調べ，正常か否かを表示する．

```
#include <stdio.h>
#include <dos.h>
```

```
void main(void)
{
      union REGS   inregs;
      union REGS   outregs;

      inregs.h.ah = 0x40;
      int86(0x67, &inregs, &outregs);
      if (outregs.h.ah == 0) {
            printf("EMMマネージャは正常です\n");
      } else {
            printf("EMSマネージャは異常です\n");
      }
}
```

2 ページフレームセグメントの取得 EMS

割り込み INT　67H

入　力 AH←41H

出　力 AH→ステータスコード（表3-32：p.207参照）
BX→ページフレームセグメント

解　説 ページフレームのセグメントアドレスを取得します．

サンプル ページフレームセグメントを取得し，そこにポインタ型変数（*data）を割り付けます．この例では，dataは64Kバイトの大きさの文字型配列として使用でき（連続して4ページ確保している場合），マッピングを変更することによって，さらに巨大な配列として利用することも可能になります．

```
#include <stdio.h>
#include <dos.h>

void main(void)
{
      union REGS   inregs;
      union REGS   outregs;
      int PageFrameSegment;
      unsigned char far *data;

      inregs.h.ah = 0x41;
      int86(0x67, &inregs, &outregs);
      PageFrameSegment = outregs.x.bx;
      data = (unsigned char far *)MK_FP(PageFrameSegment, 0);
            /* ページフレームセグメントをdataに割り付ける */
            /* なお，このdataを変数と使用する前に，マッピ */
            / * ングをしなければならない.              */
            /* [5]ハンドルページのマップ，アンマップを参照*/
      }
}
```

3 未アロケートページカウントの取得 EMS

割り込み INT 67H

入　力 AH←42H

出　力 AH→ステータスコード（表3-32：p.207参照）
BX→未アロケートページ数
DX→総ページ数

解　説 EMSの総ページ数および，利用されていないページ数を取得します．

サンプル EMSの総ページ数と未使用ページ数を取得し，表示します．

```
#include <stdio.h>
#include <dos.h>

void main(void)
{
        union REGS   inregs;
        union REGS   outregs;

        inregs.h.ah = 0x42;
        int86(0x67, &inregs, &outregs);
         printf("EMS page free/total : %d / %d\n", outregs.x.bx,
outregs.x.dx);
}
```

4 ページの割り当て EMS

割り込み NT 67H

入　力 AH←43H
BX←割り当てるページ数

出　力 AH→ステータスコード（表3-32：p.207参照）
DX→EMMハンドル

解　説 BXに示したページ数のEMSを確保します．DXレジスタには，EMMドライバが割り当てたハンドル番号が返されます．

サンプル EMSメモリを8ページ（128Kバイト）確保します．このとき，EMMハンドルを取得し，その番号を表示します．

```
#include <stdio.h>
#include <dos.h>

void main(void)
{
        union REGS   inregs;
        union REGS   outregs;
        int  EMMHandle;

        inregs.h.ah  =  0x43;
        inregs.x.bx  =  0x8;              /*  確保するEMSページ数 */
        int86(0x67, &inregs, &outregs);
        EMMHandle  =  outregs.x.dx;       /*  取得したEMMハンドル  */
     printf("EMSメモリを %d ページ確保しました (handle %d)", inregs.x.bx, EMMHandle);
}
```

5 ハンドルページのマップ，アンマップ EMS

割り込み INT 67H

入　力 AH←44H
AL←物理ページ番号
BX←論理ページ番号
DX←EMMハンドル

出　力 AH→ステータスコード（表3-32：p.207参照）

解　説 物理ページ（一般に64Kバイトの連続した空間を4分割した，0～3の4ページ）に論理ページを割り当てます．

サンプル

```
#include <stdio.h>
#include <dos.h>

void main(void)
{
        union REGS   inregs;
        union REGS   outregs;
        int  EMMHandle;
        int  PhysicalPage,  LogicalPage,  PageFrameSegment;

     /* ここで［4］ページの割り当てを実行し，EMMハンドルを取得しておく   */

     /* ページのマッピング（論理ページ0～3）（ページの割り当て）          */
        for(PhysicalPage = 0; PhysicalPage <= 3; PhysicalPage++) {
                LogicalPage  =  PhysicalPage;
                inregs.h.ah  =  0x44;
                inregs.h.al  =  PhysicalPage;
                inregs.x.bx  =  LogicalPage;
                inregs.x.dx  =  EMMHandle;
```

```
                int86(0x67, &inregs, &outregs);
            }

        /* ここでEMSにデータを転送する．[2]ページフレームセグメントの取得を */
        /* 実行し，そのアドレスにデータを転送することによって，実現される    */

        /* ページのマッピング（論理ページ4～7）（ページの切り替え）           */
            for(PhysicalPage = 0; PhysicalPage <= 3; PhysicalPage++) {
                LogicalPage = PhysicalPage + 4;/*論理ページ=物理ページ+4*/
                inregs.h.ah = 0x44;
                inregs.h.al = PhysicalPage;
                inregs.x.bx = LogicalPage;
                inregs.x.dx = EMMHandle;
                int86(0x67, &inregs, &outregs);
            }

        /* ここで再度EMSにデータを転送する．以後，論理ページを0～3／4～7と切 */
        /* り替えることによって，前回転送したデータと今回のデータを切り替えて   */
        /* 呼び出すことができる                                              */
}
```

6 ページの開放 EMS

割り込み INT 67H

入力 AH←45H
DX←EMMハンドル

出力 AH→ステータスコード（表3-32：p.207参照）

解説 DXで示したハンドルが確保しているページを開放します．

サンプル 確保したページを開放します．当然のことながら，あらかじめページを確保していなければ無意味です．EMMHandleは確保時に取得したものを設定しておいてください．

```
#include <stdio.h>
#include <dos.h>

void main(void)
{
        union REGS  inregs;
        union REGS  outregs;
        int EMMHandle;

        /* EMMHandleは，[2]ページの割り当て時に取得したものを設定する     */
        EMMHandle = 1;   /* ここでは例として1 */
        inregs.h.ah = 0x45;
        inregs.x.dx = EMMHandle;
        int86(0x67, &inregs, &outregs);
}
```

7 バージョンの取得 EMS

割り込み INT 67H

入　力 AH←46H

出　力 AH→ステータスコード（表3-32：p.207参照）
AL→バージョン番号

解　説 ALレジスタに，EMMドライバのバージョンが返されます．上位4ビットが，整数部，下位4ビットが，小数部です．

サンプル EMSのバージョンを取得し，表示します．

```
#include <stdio.h>
#include <dos.h>

void main(void)
{
        union REGS   inregs;
        union REGS   outregs;

        inregs.h.ah = 0x46;
        int86(0x67, &inregs, &outregs);
        printf("This EMS version is %x ", outregs.h.al);
}
```

8 ページマップのセーブ EMS

割り込み INT 67H

入　力 AH←47H
DX←EMMハンドル

出　力 AH→ステータスコード（表3-32：p.207参照）

解　説 マッピング情報を内部エリアにセーブします．このファンクションは，EMS3.xと互換を取るためにあるものなので，4.xを前提にプログラムを組む場合は，使用しないようにしてください．

9 ページマップの取得

割り込み INT 67H

入力 AH←48H
DX←EMMハンドル

出力 AH→ステータスコード（表3-32：p.209参照）

解説 ページマップのセーブで格納されたデータを元に，マッピングを復活します．このファンクションも3.xとの互換のために用意されているものです．

10 ハンドル数の取得 EMS

割り込み INT 67H

入力 AH←4BH

出力 AH→ステータスコード（表3-32：p.207参照）
BX→オープンしているハンドル数

解説 オープンされているEMMのハンドル数をBXレジスタに返します．システムで利用する，ハンドル0を含みます．

サンプル 現在使用されているハンドル数を取得し，表示します．

```
#include  <stdio.h>
#include  <dos.h>

void  main(void)
{
        union  REGS   inregs;
        union  REGS   outregs;

        inregs.h.ah  =  0x4b;
        int86(0x67,  &inregs,  &outregs);
    printf("現在使用中のハンドル数は %d (システム用のハンドルを含む)", outregs.x.bx);
}
```

11 ハンドルページの取得 EMS

割り込み INT 67H

入力 AH←4CH

DX←EMMハンドル

出　力　**AH**→ステータスコード（表3-32：p.207参照）

BX→割り当てられている論理ページ数

解　説　DXレジスタで指定したハンドルに割り当てられている論理ページ数をBXレジスタに返します．

サンプル　EMMハンドル1に割り当てられている論理ページ数を取得し，表示します．このプログラム実行に先立って，EMMハンドル1が使用されていなければなりません（FEPなどをEMSに組み込んでいる場合は，それらがEMMハンドル1になります）．

```
#include <stdio.h>
#include <dos.h>

void main(void)
{
        union REGS   inregs;
        union REGS   outregs;
        int  EMMHandle;

        EMMHandle=1;                        /* 例としてEMMハンドル1を使用します */

        inregs.h.ah  =  0x4c;
        inregs.x.dx  =  EMMHandle;
        int86(0x67,  &inregs,  &outregs);
    printf("EMMハンドル%d の使用している論理ページは%d", EMMHandle, outregs.x.bx);
}
```

12 全ハンドルページの取得 EMS

割り込み　**INT　67H**

入　力　**AH**←**4DH**

ES：DI←ページ数情報を書き込むアドレス

出　力　**AH**→ステータスコード（表3-32：p.207参照）

BX→オープンしているハンドル数

解　説　BXレジスタに，オープンしているハンドル数を返し，各ハンドルに割り当てられているページ数をES：DIで示されたアドレスを先頭に，書き込みます．EMSハンドル1ワード＋ページ数1ワードの，計4バイト一組のデータとして書き込まれます．これを考慮して，データ領域を確保する必要があります．

なお，ES：DIで示されるアドレスには，以下のようなデータが書き込まれます．EMMハンドル番号と，ページ数が一組となったデータのくり返しになります．

0000H	EMMハンドル番号
0002H	割り当てられているページ数
0004H	EMMハンドル番号
0006H	割り当てられているページ数
0008H ・ ・ ・	・ ・ ・

サンプル

```
#include <stdio.h>
#include <dos.h>

unsigned int data[2 * 20];  /* 最大20個分のデータ領域を用意します */
                            /* 必要に応じて増やしてください */

void main(void)
{
        union REGS   inregs;
        union REGS   outregs;
        struct SREGS segregs;
        unsigned int i;

        segregs.es = FP_SEG(data);
        inregs.x.di = FP_OFF(data);
        inregs.h.ah = 0x4d;
        int86x(0x67, &inregs, &outregs, &segregs);
        printf("オープンしているハンドル数=%u\n", outregs.x.bx);
        for(i = 0; i < outregs.x.bx; i++) {
                printf("ハンドル番号=%u\n", data[i * 2]);
                printf("割り当てられているページ数=%u\n", data[i * 2 + 1]);
        }
}
```

13 ページマップの取得 EMS

割り込み **INT 67H**

入力 **AH←4EH**
AL←00H
ES：DI←マッピング情報を書き込むアドレス

出力 **AH**→ステータスコード（表3-32：p.207参照）

解　説　　すべてのマッピング情報をES：DIで示したアドレスに格納します．格納に必要なバッファのサイズは，ページマップ格納配列のサイズ取得ファンクションで求めてください．

14　ページマップの設定　EMS

割り込み　INT　67H

入　力　AH←4EH
AL←01H
ES：DI←マッピング情報が書き込まれているアドレス

出　力　AH→ステータスコード（表3-32：p.207参照）

解　説　　ES：DIで示したアドレスに格納されているマッピング情報を元に，マッピングを復活します．

15　ページマップの取得と設定　EMS

割り込み　INT　67H

入　力　AH←4EH
AL←02H
ES：DI←マッピング情報を書き込むアドレス
DS：SI←マッピング情報が書き込まれているアドレス

出　力　AH→ステータスコード（表3-32：p.207参照）

解　説　　ES：DIで示されたアドレスに，現在のマッピング情報を格納し，DS：SIで示されたアドレスに格納されている情報をもとに，マッピングを行います．

16　一部のマッピング情報の格納　EMS

割り込み　INT　67H

入　力　AH←4FH
AL←00H
ES：DI←マッピング情報を書き込むアドレス

出力 DS：SI←マップの一部を指定するデータが書き込まれているアドレス

解説 AH→ステータスコード（表3-32：p.207参照）

DS：SIで示されるアドレスに書き込まれているデータを元に，一部の物理ページのマッピング情報をES：DIで示されたアドレスに書き込みます．

DS：SIが示すアドレスのデータは，以下のようなフォーマットにします．

オフセット	内容
0000H	物理ページ数
0002H	物理ページに対応したセグメントアドレス
0004H	物理ページに対応したセグメントアドレス
0006H	・・・・・
・・・	

17 一部のマッピング情報の復帰 EMS

割り込み INT 67H

入力 AH←4FH
AL←01H
DS：SI←マッピング情報が書き込まれているアドレス

出力 AH→ステータスコード（表3-32：p.207参照）

解説 DS：SIで示されたアドレスに書き込まれているデータを元に，マッピングを行います．

18 一部のマッピング情報を格納する配列のサイズ取得 EMS

割り込み INT 67H

入力 AH←4FH

AL←02H

BX←部分的にマップされるページ数

出力 AH→ステータスコード（表3-32：p.207参照）

AL→配列のサイズ

解説 一部のマッピング情報の格納，復帰で利用するメモリの必要なサイズをALレジスタにバイト単位で返します．

19 複数ページのマップ，アンマップ EMS

割り込み INT 67H

入力 AH←50H

AL←00H

DX←EMMハンドル

CX←配列内のエントリ数

DS：SI←配列構造が書き込まれているアドレス

出力 AH→ステータスコード（表3-32：p.207参照）

解説 DXで示したハンドルの論理ページを物理ページにマッピングします．このとき，複数のページをマッピングすることができます．なお，論理ページにFFFFHを設定すると，対応する物理ページがアンマップされ，読み書きができなくなります．

DS：SIで示すアドレスには，論理ページ番号，物理ページ番号，それぞれ1ワードの、組みをCXで示した数だけ用意しておきます．具体的には，以下のようになります．

オフセット	内容
0000H	論理ページ番号
0002H	物理ページ番号
0004H	・ ・ ・

サンプル 複数の論理ページ(例として4つ)を物理ページに割り当てます．

```
#include <stdio.h>
```

```
#include <dos.h>

unsigned int data[2 * 4];    /* 最大4個分のデータ領域を用意します */
                             /* 必要に応じて増やしてください */
void main(void)
{
      union REGS   inregs;
      union REGS   outregs;
      struct SREGS segregs;

      data[0] = 0;     /* 物理ページ0～3に，論理ページ0～3を割り当てる設定 */
      data[1] = 0;
      data[2] = 1;
      data[3] = 1;
      data[4] = 2;
      data[5] = 2;
      data[6] = 3;
      data[7] = 3;

      segregs.ds = FP_SEG(data);
      inregs.x.si = FP_OFF(data);
      inregs.x.ax = 0x5000;
      inregs.x.dx = ???? /* ????にはハンドル数を指定 */
      inregs.x.cx = 4;   /* 4ページを一度に割り当てるので4を指定 */
      int86x(0x67, &inregs, &outregs, &segregs);
      if(outregs.h.ah) {
            printf("割り当てに失敗しました\n");
      } else {
            printf("割り当てが成功しました\n");
      }
}
```

20 複数ページのマップアンマップ EMS

割り込み INT 67H

入　力 AH←50H

AL←01H

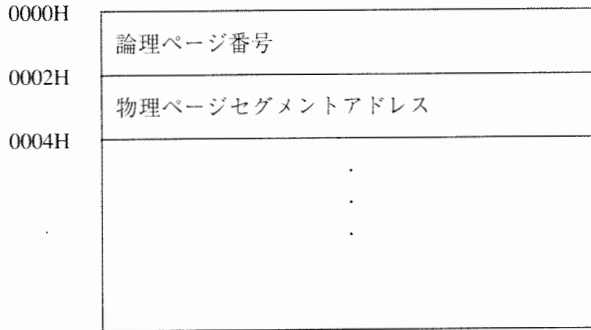
DX←EMMハンドル

CX←配列内のエントリ数

DS：SI←配列構造が書き込まれているアドレス

出　力 AH→ステータスコード（表3-32：p.207参照）

解　説 DXで示したハンドルの論理ページを物理ページにマッピングします．このとき，複数のページをマッピングすることができます．なお，論理ページにFFFFHを設定すると，対応する物理ページがアンマップされ，読み書きができなくなります．なお，このファンクションでは，物理ページを番号でなく，アドレスで指定します．

DS：SIで示すアドレスには，論理ページ番号，物理ページのアドレス，それぞれ1ワードの組みをCXで示した数だけ用意しておきます．具体的には，以下のようになります．

サンプル 複数の論理ページ(例として4つ)を物理ページに割り当てます．

```
#include <stdio.h>
#include <dos.h>

unsigned int data[2 * 4]; /* 最大4個分のデータ領域を用意します */
                           /* 必要に応じて増やしてください */

void main(void)
{
        union REGS   inregs;
        union REGS   outregs;
        struct SREGS segregs;

        data[0] = 0;     /* 物理ページ0～3に，論理ページ0～3を割り当てる設定 */
        data[1] = 0xc000;
        data[2] = 1;
        data[3] = 0xc400;
        data[4] = 2;
        data[5] = 0xc800;
        data[6] = 3;
        data[7] = 0xcc00;

        segregs.ds = FP_SEG(data);
        inregs.x.si = FP_OFF(data);
        inregs.x.ax = 0x5001;
        inregs.x.dx = ???? /* ????にはハンドル数を指定 */
        inregs.x.cx = 4;   /* 4ページを一度に割り当てるので4を指定 */
        int86x(0x67, &inregs, &outregs, &segregs);
        if(outregs.h.ah) {
                printf("割り当てに失敗しました\n");
        } else {
                printf("割り当てが成功しました\n");
        }
}
```

21　ページの再割り当て

割り込み　INT　67H

入　力　AH←51H
DX←EMMハンドル
BX←再割り当てのページ数

出　力　AH→ステータスコード（表3-32：p.207参照）
BX→再割り当てされたページ数

解　説　DXレジスタで示したハンドルの論理ページ数を増やしたり，減らしたりします．BXレジスタに，必要なページ数を指定します．BXには，増やしたり減らしたりする分を指定するのではなく，全体で何ページかを指定します．現在割り当てられているページ数よりもBXが大きければその差分を割り当て，現在割り当てられているページ数よりもBXが小さければその差分を開放します．

22　ハンドル属性の取得

割り込み　INT　67H

入　力　AH←52H
AL←00H
DX←EMMハンドル

出　力　AH→ステータスコード（表3-32：p.207参照）
AL→0：ハンドルは揮発性
1：ハンドルは不揮発性

解　説　DXで指定したハンドルの属性を得ます．

23　ハンドルの属性の設定 EMS

割り込み　INT　67H

入　力　AH←52H
AL←01H
DX←EMMハンドル

BL←0：ハンドルの属性を揮発性にする

1：ハンドルの属性を不揮発性にする

出力 AH→ステータスコード（表3-32：p.207参照）

解説 DXで指定したハンドルの属性をBLで示した属性に変更します．

24 不揮発性属性のサポート可能性の調査 EMS

割り込み INT 67H

入力 AH←52H

AL←02H

出力 AH→ステータスコード（表3-32：p.207参照）

AL→0：揮発性のみサポート

1：揮発性，不揮発性両方サポート

解説 不揮発性のサポートを行っているかどうか調べます．一般に，386以上のCPUで仮想86モードを利用して実現しているEMMマネージャは不揮発性をサポートしていません．周辺機器メーカーから発売されている，286以下のCPUのマシン用のハードウェアEMSボードの中には，不揮発性をサポートしているものがあります．

25 ハンドル名の取得 EMS

割り込み INT 67H

入力 AH←53H

AL←00H

DX←EMMハンドル

ES：DI←ハンドル名を書き込むアドレス

出力 AH→ステータスコード（表3-32：p.207参照）

解説 DS：DIで示したアドレスに，DXで示したEMMハンドルのハンドル名を返します．ハンドル名は8バイトです．

サンプル ハンドル名を取得し，表示します．

```
#include <stdio.h>
#include <dos.h>
```

```
char name[8];

void main(void)
{
        union REGS   inregs;
        union REGS   outregs;
        struct SREGS segregs;
        unsigned int i;

        segregs.es  = FP_SEG(name);
        inregs.x.di  = FP_OFF(name);
        inregs.h.ah  = 0x53;
        inregs.h.al  = 0;
        inregs.x.dx = 1;     /* ハンドル名を取得するハンドル番号(例として1) */
        int86x(0x67, &inregs, &outregs, &segregs);
        printf("ハンドル名=%s\n", name);
}
```

26 ハンドル名の設定 EMS

割り込み INT 67H

入力 AH←53H

AL←01H

DX←EMMハンドル

ES：SI←ハンドル名が書き込まれているアドレス

出力 AH→ステータスコード（表3-32：p.207参照）

解説 DXで示したEMMハンドルに，ES：SIで示したアドレスに格納されているハンドル名を設定します．ハンドル名は8バイトです．

サンプル ハンドル名を設定します．ここではハンドル名を“TEST”とします．

```
#include <stdio.h>
#include <dos.h>

void main(void)
{
        union REGS   inregs;
        union REGS   outregs;
        struct SREGS segregs;
        char HandleName[] = "TEST";          /* ハンドル名 */
        int EMMHandle;

        /*   ここで必ずページ割当を行う：[4]ページの割り当てを参照のこと  */

        EMMHandle = 1; /* ここでは例としてハンドルは1 */
        inregs.x.ax = 0x5301;
```

```
	inregs.x.dx	=  EMMHandle;		/* EMMHandleはページ割当時に取
得したもの  */
	inregs.x.si	=  FP_OFF(HandleName);	/* ハンドル名のオフセット */
	segregs.ds	=  FP_SEG(HandleName);	/* ハンドル名のセグメント */
	int86x(0x67, &inregs, &outregs, &segments);
}
```

27 ハンドルのディレクトリ情報の取得 EMS

割り込み INT 67H

入　力 AH←54H
AL←00H
ES：DI←ディレクトリ情報を書き込むアドレス

出　力 AH→ステータスコード（表3-32：p.207参照）
AL→ハンドル数

解　説 すべてのオープンされているハンドルの値と，ハンドルに割り当てられている名前をES：DIで示したアドレスに書き込みます．データフォーマットは，以下のようになり，ハンドルの値1ワードと，ハンドル名8バイトのくり返しです．なお，ハンドル名が割り当てられていなければ，ハンドル名の領域は，NULLで埋められます．

オフセット	内容
0000H	ハンドルの値
0002H	ハンドル名
000AH	・ ・ ・

28 指定の名前を持つハンドルの検索 EMS

割り込み INT 67H

入　力 AH←54H
AL←01H

出力 ES：SI←ハンドル名を書き込んであるアドレス

出力 AH→ステータスコード（表3-32：p.207参照）
DX→ハンドルの値

解説 DS：SIで示されたアドレスに格納されているハンドル名を持つハンドルの番号を返します．

29 ハンドル総数の取得 EMS

割り込み INT 67H

入力 AH←54H
AL←02H

出力 AH→ステータスコード（表3-32：p.207参照）
BX→ハンドルの総数

解説 EMMドライバがサポートしているハンドルの総数をBXレジスタに返します．

サンプル ハンドルの総数を取得して表示します

```
#include <stdio.h>
#include <dos.h>

void main(void)
{
    union REGS  inregs;
    union REGS  outregs;

    inregs.x.ax = 0x5402;
    int86(0x67, &inregs, &outregs);
    if(outregs.h.ah) {
    printf("ハンドル総数の取得に失敗しました\n");
    } else {
        printf("ハンドル総数=%u\n", outregs.x.bx);
    }
}
```

30 ページマップの変更とジャンプ EMS

割り込み INT 67H

入力 AH←55H
AL←00H：ページ番号

01H：セグメント

DX←EMMハンドル

DS：SI←ジャンプアドレスなどを含むデータが書き込まれているアドレス

出　力 **AH**→ステータスコード（表3-32：p.207参照）

解　説 マッピングを変更し，さらに，指定のアドレスにFARジャンプします．DS：SIで示されたアドレスに書き込むデータは，次のような形式になっています．

オフセット	内容
0000H	ジャンプ先のアドレス
0004H	マッピング情報のエントリ数
0005H	マッピング情報のアドレス
0009H	

ここで，マッピング情報のアドレスで示されるアドレスには，次のような形式でデータを格納します．マッピング情報のエントリ数の分，論理ページ番号（1ワード），物理ページ番号（セグメントアドレス）（1ワード）のデータの組みを繰り返します．

オフセット	内容
0000H	論理ページ番号
0002H	物理ページ番号または，セグメントアドレス
0004H	・ ・ ・

31 ページマップの変更とコール EMS

割り込み **INT　67H**

入　力 **AH**←**56H**

AL←00H：ページ番号

01H：セグメント

DX←EMMハンドル

DS：SI←ターゲットアドレスのデータが書き込まれているアドレス

出力 AH→ステータスコード（表3-32：p.207参照）

解説 DS：SIで示されたアドレスに書き込まれているデータに従い，ページマップの変更を行い，指定したアドレスにFARコールします．DS：SIで示されるアドレスには，次のような形式でデータを格納しておきます．

オフセット	内容
0000H	コール先のアドレス
0004H	新しいマッピング情報のエントリ数
0005H	新しいマッピング情報のアドレス
0009H	現在のマッピング情報のエントリ数
000AH	現在のマッピング情報のアドレス
000FH	

ここで，マッピング情報のアドレス（新しいアドレス，現在のアドレスともに）で示されるアドレスには，次のような形式でデータを格納します．マッピング情報のエントリ数の分，論理ページ番号（1ワード），物理ページ番号（セグメントアドレス）（1ワード）のデータの組みを繰り返します．

オフセット	内容
0000H	論理ページ番号
0002H	物理ページ番号または，セグメントアドレス
0004H	・ ・ ・

32 ページマップの変更に必要なスタックサイズの取得 EMS

割り込み INT 67H

入力 AH←56H
AL←02H

出力 AH→ステータスコード（表3-32：p.207参照）
BX→必要とするスタックサイズ

解　説	ページマップの変更とジャンプおよび，ページマップの変更とコールで必要とするスタックサイズをバイト単位で，BXレジスタに返します．

33　メモリ領域のコピー　EMS

割り込み　INT　67H

入　力　AH←57H
AL←00H
DS：SI←コピー領域の指定などのデータが格納されているアドレス

出　力　AH→ステータスコード（表3-32：p.207参照）

解　説　DS：SIで示されたアドレスに格納されているデータを元に，メモリ間のコピーを行います．DS：SIで示されるアドレスには，次のような形式で，データを格納しておきます．

オフセット	内容
0000H	コピーするバイト数
0004H	コピー元のメモリタイプ＊
0005H	コピー元のハンドル番号(内部メモリ：0)
0007H	コピー元の先頭オフセット
0009H	コピー元の論理ページまたは，セグメント
000BH	コピー先のメモリタイプ＊
000CH	コピー先のハンドル番号(内部メモリ：0)
000EH	コピー先の先頭オフセット
0010H	コピー先の論理ページまたは，セグメント
0012H	

＊メモリタイプ
　0：内部メモリ
　1：拡張メモリ

サンプル　テキストVRAMの内容をEMSにコピーします．

```
#include <stdio.h>
#include <dos.h>
```

```
struct param {     /* データ格納用構造体 */
      unsigned int copybyte;
      char moto_type;
      unsigned int moto_handle;
      unsigned int moto_off;
      unsigned int moto_page;
      char saki_type;
      unsigned int saki_handle;
      unsigned int saki_off;
      unsigned int saki_page;
};

void main(void)
{
      union REGS   inregs;
      union REGS   outregs;
      struct SREGS segregs;
      struct param data;

      data.copybyte = 4000;    /* コピーするバイト数 */
      data.moto_type = 0;      /* コピー元は内部メモリ */
      data.moto_handle = 0;    /* 内部メモリなので0 */
      data.moto_off = 0;       /* コピー元オフセットは0 */
      data.moto_page = 0xa000; /* 内部メモリなのでセグメント指定 */
      data.saki_type = 1;      /* コピー先はEMS */
      data.saki_handle = 2;    /* ハンドルは例として2(適宜変更してください) */
      data.saki_off = 0;       /* コピー先オフセットは0 */
      data.saki_page = 0;      /* コピー先論理ページは0 */

      inregs.x.ax = 0x5700;
      segregs.ds = FP_SEG(&data);
      inregs.x.si = FP_OFF(&data);
      int86x(0x67, &inregs, &outregs, &segregs);
      if(outregs.h.ah) {
      printf("コピーに失敗しました\n");
      } else {
            printf("コピーが正常に終了しました\n");
      }
}
```

34 メモリ領域の交換 EMS

割り込み INT 67H

入　力 AH←57H

AL←01H

DS：SI←交換領域の指定などのデータが格納されているアドレス

出　力 AH→ステータスコード（表3-32：p.207参照）

解　説　DS：SIで示されたアドレスに格納されているデータを元に，メモリの入れ替えを行います．DS：SIで示されるアドレスには次のようなデータを格納しておきます．

オフセット	内容
0000H	交換するバイト数
0004H	交換元のメモリタイプ*
0005H	交換元のハンドル番号(内部メモリ：0)
0007H	交換元の先頭オフセット
0009H	交換元の論理ページ(内部メモリの場合はセグメント)
000BH	交換先のメモリタイプ*
000CH	交換先のハンドル番号(内部メモリ：0)
000EH	交換先の先頭オフセット
0010H	交換先の論理ページ(内部メモリの場合はセグメント)
0012H	

＊メモリタイプ
　0：内部メモリ
　1：拡張メモリ

サンプル　テキストVRAMの内容とEMSに格納されているデータを交換します

```
#include <stdio.h>
#include <dos.h>

struct param {      /* データ格納用構造体 */
       unsigned int changebyte;
       char moto_type;
       unsigned int moto_handle;
       unsigned int moto_off;
       unsigned int moto_page;
       char saki_type;
       unsigned int saki_handle;
       unsigned int saki_off;
       unsigned int saki_page;
};

void main(void)
{
       union REGS   inregs;
       union REGS   outregs;
       struct SREGS segregs;
       struct param data;

       data.changebyte = 4000; /* 交換するバイト数 */
```

```
data.moto_type = 0;       /* 交換元は内部メモリ */
data.moto_handle = 0;     /* 内部メモリなので0 */
data.moto_off = 0;        /* 交換元オフセットは0 */
data.moto_page = 0xa000;/* 内部メモリなのでセグメント指定 */
data.saki_type = 1;       /* 交換先はEMS */
data.saki_handle = 2;     /*ハンドルは例として2(適宜変更してください)*/
data.saki_off = 0;        /* 交換先オフセットは0 */
data.saki_page = 0;       /* 交換先論理ページは0 */

inregs.x.ax = 0x5701;
segregs.ds = FP_SEG(&data);
inregs.x.si = FP_OFF(&data);
int86x(0x67, &inregs, &outregs, &segregs);
if(outregs.h.ah) {
printf("交換に失敗しました\n");
} else {
     printf("交換が正常に終了しました\n");
}
}
```

35 マップ可能な物理アドレス配列の取得 EMS

割り込み INT 67H

入　力 AH←58H
AL←00H
ES：DI←データを書き込むアドレス

出　力 AH→ステータスコード（表3-32：p.207参照）
CX→物理ページのエントリ数

解　説 ES：DIで示されたアドレスに，マップ可能な物理ページと，その物理ページに対応したセグメントの情報を書き込みます．物理ページのセグメント，物理ページ番号がそれぞれ1ワードずつCXで示された回数繰り返されるようなデータ構造になります．具体的には，次のようになっています．

オフセット	内容
0000H	物理ページのセグメント
0002H	物理ページ番号
0004H	・ ・ ・

36 マップ可能な物理アドレス配列のエントリ数の取得 EMS

割り込み　INT　67H

入　力　AH←58H
AL←01H

出　力　AH→ステータスコード（表3-32：p.207参照）
CX→物理ページのエントリ数

解　説　CXレジスタに，マップ可能な物理ページの数を返します．

37 ハードウェア構成に関する情報の取得 EMS

割り込み　INT　67H

入　力　AH←59H
AL←00H
ES：DI←拡張メモリのハードウェア構成のデータを書き込むアドレス

出　力　AH→ステータスコード（表3-32：p.207参照）

解　説　ES：DIで示されるアドレスに，ハードウェア構成のデータを書き込みます．書き込まれるデータの形式は，以下のようになっています．

オフセット	内容
0000H	ROW物理ページのサイズ
0002H	代替マッピングレジスタの数
0004H	マッピング情報の格納に必要なバイト数
0004H	DMA chに割り当てられるレジスタセットの数
0004H	DMAレジスタセットの利用モード*
0006H	・ ・ ・

*0：代替マップ可能
　1：DMAレジスタは1個

なお，ROW物理ページのサイズとは，EMSがサポートする標準より小さいページサイズのことをいいます．ROWページのサイズは16バイト単位で返されます．EMSの標準ページサイズは16KBなので，ここには400H以下の値が返ってきます．しかしながら，98では一般に標準サイズよりページサイズを小さくすることができずROWページのサイズには400Hが返ってきます．

38 未アロケートのROWページ数の取得 EMS

割り込み INT 67H

入力 AH←59H
AL←01H

出力 AH→ステータスコード（表3-32：p.207参照）
BX→未アロケートのROWページ数
DX→ROWページの総数

解説 BXレジスタに，未アロケートのROWページ数，DXレジスタにROWページの総数を返します．ROWページですので，必ずしも標準サイズではありません．

39 標準サイズのページの割り当て EMS

割り込み INT 67H

入力 AH←5AH
AL←00H
BX←割り当てるページ数

出力 AH→ステータスコード（表3-32：p.207参照）
DX→EMMハンドル

解説 BXで示したページ数のページを割り当てます．「ページの割り当て」のファンクションとは，0ページが割り当てられる点で違います．

40 ROWページの割り当て EMS

割り込み INT 67H

入力 AH←5AH

AL←01H

BX←割り当てるROWページ数

出　　力　AH→ステータスコード（表3-32：p.207参照）

DX→EMMハンドル

解　　説　BXで示したページ数のROWページを割り当てます．0ページも割り当てることが可能です．

41 代替マップレジスタセットの取得 EMS

割り込み　INT　67H

入　　力　AH←5BH

AL←00H

出　　力　AH→ステータスコード（表3-32：p.207参照）

BL→マップレジスタセットの番号

ES：DI→データを書き込まれているアドレス

解　　説　その時点でアクティブになっているマップレジスタによって，返されるデータは変わります．「代替マップレジスタセットの設定」における，BLレジスタに返される値で判断します．

「代替マップレジスタセットの設定」でBLレジスタに返される値が0の場合現在のマッピング情報をES：DIで示されるアドレスに返します．

「代替マップレジスタセットの設定」でBLレジスタに返される値が0以外の場合現時点で使用されている代替マップレジスタセットの番号が，BLレジスタに返されます．

42 代替マップレジスタセットの設定 EMS

割り込み　INT　67H

入　　力　AH←5BH

AL←01H

BL←新しいマップレジスタセットの番号

ES：DI←マッピング情報を格納しているアドレス

出　　力　AH→ステータスコード（表3-32：p.207参照）

解　説　BLレジスタに0を指定した場合，ES：DIの内容が0でなければ，そのマッピング情報をマッピングレジスタにコピーされる．0であればなにも行われない．
BLレジスタに0以外を指定した場合は，指定のマップレジスタがアクティブになります．

43 代替マップセーブ配列のサイズ取得 EMS

割り込み　INT　67H

入　力　AH←5BH
AL←02H

出　力　AH→ステータスコード（表3-32：p.207参照）
DX←配列のサイズ

解　説　「代替マップレジスタセットの設定」での，マッピング情報を格納するために必要な領域のサイズをDXレジスタに返します．

44 代替マップレジスタセットの割り当て EMS

割り込み　INT　67H

入　力　AH←5BH
AL←03H

出　力　AH→ステータスコード（表3-32：p.207参照）
BL←マップレジスタセットの番号

解　説　使用できる代替マップレジスタセットの番号をBLレジスタに返します．代替マップレジスタがサポートされていなければ，0が返されます．

45 代替マップレジスタセットの開放 EMS

割り込み　INT　67H

入　力　AH←5BH
AL←04H
BL←マップレジスタセットの番号

出　力　AH→ステータスコード（表3-32：p.207参照）

解　説　BLで示されたマップレジスタセットを開放します．

46 DMAレジスタセットの割り当て EMS

割り込み　INT　67H

入　力　AH←5BH
AL←05H

出　力　AH→ステータスコード（表3-32：p.207参照）
BL→DMAレジスタセットの番号

解　説　DMAレジスタセットの番号をBLレジスタに返します．DMAレジスタセットがサポートされていない場合は，0が返されます．

47 代替マップレジスタによるDMAの使用許可 EMS

割り込み　INT　67H

入　力　AH←5BH
AL←06H
BL←代替マップレジスタセットの番号
DL←DMAチャネル番号

出　力　AH→ステータスコード（表3-32：p.207参照）

解　説　BLレジスタで指定された代替マップレジスタセットを通じて，DLレジスタで指定されたDMAチャネルでのDMAアクセスを可能にします．

48 代替マップレジスタに対するDMAの使用禁止 EMS

割り込み　INT　67H

入　力　AH←5BH
AL←07H
BL←代替マップレジスタセットの番号

出　力	AH→ステータスコード（表3-32：p.207参照）
解　説	BLレジスタで指定された代替マップレジスタセットに対するDMAチャネルへのアクセスを禁止します．

49 DMAレジスタセットの開放

割り込み	INT　67H
入　力	AH←5BH AL←08H
出　力	AH→ステータスコード（表3-32：p.207参照） BL→DMAレジスタセットの番号
解　説	DMAレジスタセットを開放します．BLレジスタに，使用できなくなるDMAレジスタセットの番号が返されます．

50 ウォームブートのための拡張メモリハードウェアの準備 EMS

割り込み	INT　67H
入　力	AH←5CH
出　力	AH→ステータスコード（表3-32：p.207参照）
解　説	ウォームブートのための拡張メモリハードウェアの準備をします．拡張メモリのハードウエアは初期化されます．

51 OS/Eファンクションセットの使用許可の設定 EMS

割り込み	INT　67H
入　力	AH←5DH AL←00H BX，CX←アクセスキー
出　力	AH→ステータスコード（表3-32：p.207参照） BX，CX→アクセスキー

解　説　OS／E指定のファンクションを，すべてのプログラムが使用できるように許可します．

52 OS／Eファンクションセットの使用禁止の設定 EMS

割り込み INT　67H

入　力 AH←5DH
AL←01H
BX，CX←アクセスキー

出　力 AH→ステータスコード（表3-32：p.207参照）
BX，CX→アクセスキー

解　説　OS／E指定のファンクションをOS／E以外のプログラムが使用することを禁止します．

53 アクセスキーのリターン EMS

割り込み INT　67H

入　力 AH←5DH
AL←02H
BX，CX←アクセスキー

出　力 AH→ステータスコード（表3-32：p.207参照）

解　説　OS／EがEMMにアクセスキーを返せるようにします．

54 ページフレーム用のバンクのステータス取得 EMS

割り込み INT　67H

入　力 AH←70H
AL←00H

出　力 AH→ステータスコード（表3-32：p.207参照）
AL→0：ページフレームに使用可
1：ページフレームに使用不可

ページフレーム用のバンクが，ページフレームとして使えるかどうか調べます．このファンクションは，NECのDOSに付属するEMMドライバ独特のもので，VRAMの裏をEMSのページフレームとして使用する場合のみ有効です．このファンクションを持つEMMドライバは一般には存在しません．

55 ページフレーム用バンクの状態の設定 EMS

割り込み INT　67H

入　力 AH←70H

AL←01H

BL←0：ページフレーム

1：VRAM

出　力 AH→ステータスコード（表3-32：p.207参照）

AL→0：ページフレームに使用可

1：ページフレームに使用不可

解　説 BLレジスタで指定された状態にページフレーム用のバンクを切り替えます．このファンクションは，NECのDOSに付属するEMMドライバ独特のもので，VRAMの裏をEMSのページフレームとして使用する場合のみ有効です．このファンクションを持つEMMドライバは一般には存在しません．

■サンプルプログラム

```
/* EMSを使い，テキストVRAMの内容の待避，復活を行います
   このプログラムでは，テスト画面を表示したのち，その内容を
   EMSに待避します．その後，画面消去された状態で，EMS
   からデータをテキストVRAMに書き戻し，もとの画面を表示
   します．                                              */

#include <stdio.h>
#include <stdlib.h>

int pageget(int);
int map(int);
unsigned int getseg(void);
void emm_rw(unsigned int, int);
void emm_free(int);

void main(void)
{
```

```
	int i, ha;
	char far *emmtest, *emm = "EMMXXXX0";
	unsigned int dseg, pageseg;

	/* int 67h のベクタ取得 */
	asm {
		mov ax,3567h
		int 21h
		mov dseg,es
	}
	emmtest = (char far*)((long)dseg * 0x10000 + 0x0a);

	/* EMMXXXX0 という文字列をチェックすることにより,
	   EMMドライバの存在をチェック                          */
	for(i = 0; i < 8; i++) {
		if(*(emmtest + i) != *(emm + i)) {
			printf("EMMドライバは存在しません\n");
			return;
		}
	}
	if((ha = pageget(1)) == -1) {
		printf("ページが確保できません\n");
	}
	if(map(ha) == -1) {
		printf("ページがマップできません\n");
		emm_free(ha);
		exit(-1);
	}
	if((pageseg = getseg()) == 0xffff) {
		printf("セグメントの取得に失敗しました\n");
		emm_free(ha);
		exit(-1);
	}
	printf("\x1b*");
	printf("emm test program\n");
	printf("EMMページセグメント = %04x\n", pageseg);
	for(i = 0; i < 20; i++) {
		printf("test\n");
	}
	printf("何かキーを押すと画面をEMSに待避し, 消去します\n");
	getch();
	emm_rw(pageseg, 0);
	printf("\x1b*");
	printf("何かキーを押すと画面を復活します\n");
	getch();
	emm_rw(pageseg, 1);
	emm_free(ha);
}

/* ページの割り当て */
int pageget(int page)
{
	int ha;
	unsigned char status;
```

```
        /*  ページ割り当てファンクションの呼び出し  */
        /*  page  で指定した数のページを割り当てます  */
        asm  {
                mov  bx,page
                mov  ah,43h
                int  67h
                mov  status,ah
                mov  ha,dx
        }
        if(status  !=  0)  {
                return  -1;
        } else  {
                return  ha;
        }
}

/*  ページのマップ  */
int  map(int  ha)
{
        unsigned  char  status;

        /*  ページマップファンクションの呼び出し  */
        /*  物理ページ0を論理ページ0に割り当てます  */
        asm  {
                mov  dx,ha
                mov  ax,4100h
                mov  bx,0
                int  67h
                mov  status,ah
        }
        if(status  !=  0)  {
                printf("%02x\n",  status);
                return  -1;
        }
        return  0;
}

/*  ページフレームのセグメントの取得  */
unsigned  int  getseg(void)
{
        unsigned  int  pageseg;
        unsigned  char  status;

        /*  ページフレームセグメントの取得  */
        asm  {
                mov  ah,41h
                int  67h
                mov  pageseg,bx
                mov  status,ah
        }
        if(status  !=  0)  {
                return  0xffff;
        } else  {
                return  pageseg;
        }
```

```
}

/*  EMS<->TVRAM */
/*  sw = 0 : TVRAM->EMS
    sw = 1 : EMS->TVRAM */
void emm_rw(unsigned int pageseg, int sw)
{
      char far *emm;
      char far *tvram;
      int i;

      /* EMSのポインタ設定 */
      emm = (char far*)(pageseg * 0x10000L);
      /* テキストVRAMのポインタ設定 */
      tvram = (char far*)0xa0000000L;

      /* EMS<->TVRAM間の転送 */
      for(i = 0; i < 4000; i++) {
            switch(sw) {
            case 0:
                  *(emm + i) = *(tvram + i);
                  break;
            case 1:
                  *(tvram + i) = *(emm + i);
                  break;
            }
      }
}

/* 確保したEMSを開放します */
void emm_free(int ha)
{
      unsigned char status;

      /* 開放ファンクション呼び出し */
      asm {
            mov dx,ha
            mov ah,45h
            int 67h
            mov status,ah
      }
      if(status != 0) {
            printf("開放に失敗しました\n");
      }
}
```

■2-8-3 XMS

XMSとは，プロテクトメモリをデータ格納領域として利用するための，ファンクションです．しかし，実際には，EMB，HMA，UMBの3つのメモリを管理するファンクションとなっています．実際，DOSにXMSドライバが含まれるようになったのはver.5からですが，ver.3.3でも，サードパーティが提供

するドライバによって実現されていました．3つのメモリは次のようなものです．

◆EMB

EMBは，いわゆるプロテクトメモリを意味します．一般にXMSファンクションというとこの領域を操作するものと解釈されることがほとんどです．DOSからは直接プロテクトメモリのアクセスはできませんので，XMSファンクションでは，メモリ転送ファンクションを使って，DOSからプロテクトメモリを操作することを実現しています．EMSと比べるとデータの格納，呼び出しのたびに，データ転送が発生するという欠点がありますが，反面，EMSのように連続したメモリ領域が最大64KBということはなく，プロテクトメモリがあるかぎり連続したメモリ空間が確保できるという利点があります．つまり，EMSのような面倒なページ切り替えは必要ないわけです．

◆HMA

HMAは，CPUがリアルモードで動作しているときに，唯一アクセス可能なプロテクトメモリ領域のことをいいます．これはプロテクトメモリの領域の最下部の64Kバイトの領域です．DOS5以降では，この領域にDOS本体を入れることができ，そうすることによって，コンベンショナルメモリの空き要領を増やすことができます．またDOS3.3以前でも，HMAをサポートするドライバを組み込むことにより，FEPなどをこの領域に置くことができます．

◆UMB

UMBは，プロテクトメモリではありません．これはリアルモードでアクセス可能な1Mバイトのメモリの中の拡張ROM領域をいいます．この空き領域にメモリを割り当て，デバイスドライバの登録などに利用されています．これにより多くのデバイスドライバを組み込んだ場合でも，コンベンショナルメモリの空き領域を増やすことができます．

●XMSファンクションの利用方法

XMSファンクションは，

①INT 2FHでドライバの確認，
②ファンクションコールアドレスの取得，
③機能コードをAHレジスタにセットし，
④取得したファンクションコールアドレスにFARコールする

ことにより呼び出します．一般のファンクションコールのように，INT ??Hを利用しない特殊な方法をとっていることに注意してください．

■XMSドライバ存在確認，コールアドレスの取得ファンクション一覧（INT　2FH）

機能コード	機能
４３Ｈ	ＸＭＳドライバの存在確認 ファンクションコールアドレスの取得

1 XMSドライバの存在確認 XMS

割り込み INT 2FH

入　力 AH←43H
AL←00H

出　力 AL←80H　　：XMSドライバが存在する
上記以外 ：XMSドライバが存在しない

解　説 XMSドライバがインストールされているか確認します．ALレジスタに80Hが返されれば，XMSドライバがインストールされていることを意味します．

サンプル XMSドライバが存在しているかいないかをチェックします．

```
#include  <stdio.h>
#include  <dos.h>

void  main(void)
{
        union  REGS   inregs;
        union  REGS   outregs;

        inregs.h.ah  =  0x43;
        inregs.h.al  =  0x00;
        int86(0x2f,  &inregs,  &outregs);
        if  (outregs.h.al  ==  0x80)  {
                printf("XMSドライバが存在します");
        }  else  {
                printf("XMSドライバは存在しません");
        }
}
```

2 XMSドライバのファンクションコールアドレスの取得 XMS

割り込み INT 2FH

入　力 AH←43H
AL←01H

出　力 ES：BX→XMSファンクションコールアドレス

解　説 XMSファンクションコールアドレスを返します．XMSファンクションを利用するには，ES：BXで示されたアドレスをFARコールします．

サンプル　XMSファンクションコールアドレスを取得します．

```
#include <stdio.h>
#include <dos.h>

void main(void)
{
        union  REGS   inregs;
        union  REGS   outregs;
        struct  SREGS  segregs;

        inregs.h.ah  =  0x43;
        inregs.h.al  =  0x01;
        int86x(0x2f,  &inregs,  &outregs,  &segregs);
        printf("アドレス=%04x:%04x\n",  segregs.es,  outregs.x.bx);
}
```

■XMSファンクション一覧

機能コード	機能
００Ｈ	ＸＭＳバージョンの取得
０１Ｈ	ＨＭＡの要求
０２Ｈ	ＨＭＡの開放
０３Ｈ	Ａ２０のグローバルな有効化
０４Ｈ	Ａ２０のグローバルな無効化
０５Ｈ	Ａ２０のローカルな有効化
０６Ｈ	Ａ２０のローカルな無効化
０７Ｈ	Ａ２０の状態の取得
０８Ｈ	ＥＭＢ空きエリア状態取得
０９Ｈ	ＥＭＢの割り当て
０ＡＨ	ＥＭＢの開放
０ＢＨ	ＥＭＢブロック転送
０ＣＨ	ＥＭＢのロック
０ＤＨ	ＥＭＢのロック解除
０ＥＨ	ＥＭＢハンドル情報の取得
０ＦＨ	ＥＭＢの再割り当て
１０Ｈ	ＵＭＢの割り当て
１１Ｈ	ＵＭＢの開放

各ファンクションの詳細を以下に示します．なお，ほとんどのファンクションにおいてAXレジスタに00Hが返された場合は，何かエラーが発生した場合です．この場合はBLレジスタにエラーコードが返されます．エラーコードの詳細は，表2-33「エラーコード一覧」を参照してください．

表2-33　エラーコード一覧

エラースコード	機能
８０Ｈ	ファンクションがインプリメントされていない
８１Ｈ	ＶＤＩＳＫインターフェースが使用されている
８２Ｈ	Ａ２０ラインエラー
８ＥＨ	ＸＭＳドライバエラー
８ＦＨ	回復不可能なＸＭＳドライバエラー
９０Ｈ	ＨＭＡ領域が存在しない
９１Ｈ	ＨＭＡ領域はすでに利用されている
９２Ｈ	ＨＭＡの割り当て要求サイズが指定より小さい
９３Ｈ	ＨＭＡ領域が割り当てられていない
９４Ｈ	Ａ２０ラインがイネーブル状態
Ａ０Ｈ	使用可能なＥＭＢがすべて割り当てられている
Ａ１Ｈ	使用可能なＥＭＢがすべて使用中
Ａ２Ｈ	ハンドルが不正
Ａ３Ｈ	転送元のハンドル値が不正
Ａ４Ｈ	転送元のオフセットが不正
Ａ５Ｈ	転送先のハンドル値が不正
Ａ６Ｈ	転送先のオフセットが不正
Ａ７Ｈ	転送する長さ指定が不正
Ａ８Ｈ	転送において不正なオーバーラップ発生
Ａ９Ｈ	パリティーチェックエラー
ＡＡＨ	ＥＭＢブロックがロックされていない
ＡＢＨ	ＥＭＢブロックがロックされている
ＡＣＨ	ＥＭＢブロックのロックカウントがオーバーフロー
ＡＤＨ	ＥＭＢブロックのロックが失敗
Ｂ０Ｈ	指定したサイズより小さいサイズのＵＭＢが使用可
Ｂ１Ｈ	使用可能なＵＭＢが存在しない
Ｂ２Ｈ	ＵＭＢのセグメント値が不正

1 XMSバージョンの取得／XMS

入力 AH←00H

出力 AH→XMSドライバのバージョンの番号（整数部）
AL→XMSドライバのバージョンの番号（小数部）
BH→XMSドライバのリビジョンの番号（整数部）
BL→XMSドライバのリビジョンの番号（小数部）
DX→1：HMAあり
0：HMAなし

解説 XMSドライバのバージョン，リビジョン番号を得ます．また，HMAが存在するか確認します．

サンプル XMSのバージョン情報を得て，表示します．

```
/*  このプログラムを実行するには，XMSドライバーが組み込まれている必要があります
   また，各サンプルに共通のget_vec()関数(XMS項目の最後に掲載)が必要です．*/
#include <stdio.h>
#include <dos.h>

void get_vec(void);

unsigned long calladd; /* コールアドレス格納用変数 */

void main(void)
{
        unsigned char vera, verb, reva, revb;
        unsigned int hma;

        get_vec();      /* ベクタ取得関数 */
        /* バージョン取得ファンクション呼び出し */
        asm {
                mov  ah,0
                call dword ptr [calladd]
                mov  vera,ah
                mov  verb,al
                mov  reva,bh
                mov  revb,bl
                mov  hma,dx
        }
        printf("バージョン=%d.%d\n", vera, verb);
        printf("リビジョン=%d.%d\n", reva, revb);
        if(hma == 0) {
                printf("HMAは存在しません\n");
        } else {
                printf("HMAは存在します\n");
        }
}
```

2 HMAの要求 /XMS

入力　**AH←01H**

DX←HMAの必要バイト数

出力　**AX**→1：割り当て成功

0：割り当て失敗

解説　DXレジスタで指定したバイト数のHMAを割り当てます．

サンプル　HMAを1Kバイト割り当てます

```
/*  このプログラムを実行するには，XMSドライバーが組み込まれている必要があります
   また，各サンプルに共通のget_vec()関数(XMS項目の最後に掲載)が必要です． */
#include <stdio.h>
#include <dos.h>

void get_vec(void);

unsigned long calladd; /* コールアドレス格納用変数 */

void main(void)
{
      unsigned int hma_byte, status;

      hma_byte = 1024;    /* 割り当てるHMAのサイズ */
      get_vec();       /* ベクタ取得関数 */
    /* バージョン取得ファンクション呼び出し */
      asm {
            mov ah,1
            mov dx,hma_byte
            call dword ptr [calladd]
            mov status,ax
      }
      if(status) {
            printf("HMAの割り当てが正常に終了しました\n");
      } else {
            printf("HMAの割り当てに失敗しました\n");
      }
}
```

3 HMAの開放 /XMS

入力　**AH←02H**

出力　**AX**→1：開放成功

0：開放失敗

解説　HMA領域を開放します．

サンプル HMAを開放します

```
/*  このプログラムを実行するには，XMSドライバーが組み込まれている必要があります
   また，各サンプルに共通のget_vec()関数(XMS項目の最後に掲載)が必要です． */
#include <stdio.h>
#include <dos.h>

void get_vec(void);

unsigned long calladd;  /* コールアドレス格納用変数 */

void main(void)
{
        unsigned int status;

        get_vec();      /* ベクタ取得関数 */
    /* バージョン取得ファンクション呼び出し */
        asm {
                mov  ah,2
                call  dword ptr [calladd]
                mov  status,ax
        }
        if(status) {
                printf("HMAの開放が正常に終了しました\n");
        } else {
                printf("HMAの開放に失敗しました\n");
        }
}
```

4 A20のグローバルな有効化／XMS

入力 AH←03H

出力 AX→1：有効化成功
0：有効化失敗

解説 A20を有効化します．

5 A20のグローバルな無効化／XMS

入力 AH←04H

出力 AX→1：無効化成功
0：無効化失敗

解説 A20を無効化します．

6 A20のローカルな有効化 /XMS

入力 AH←05H

出力 AX→1：有効化成功
0：有効化失敗

解説 A20を有効化します.

7 A20のローカルな無効化 /XMS

入力 AH←06H

出力 AX→1：無効化成功
0：無効化失敗

解説 A20を無効化します.

8 A20の状態の取得 /XMS

入力 AH←07H

出力 AX→1：A20ラインは有効
0：A20ラインは無効

解説 A20ラインの状態を取得します.

9 EMB空きエリア状態取得 /XMS

入力 AH←08H

出力 AX→EMBの最大空きメモリブロックのサイズ
DX→EMBの空きメモリのサイズ

解説 EMBの空きエリアのサイズを取得します.

サンプル EMBの空きエリアの状態を取得し表示します

```
/*  このプログラムを実行するには, XMSドライバーが組み込まれている必要があります
   また, 各サンプルに共通のget_vec()関数(XMS項目の最後に掲載)が必要です. */
#include <stdio.h>
#include <dos.h>

void get_vec(void);
```

```
unsigned long calladd; /* コールアドレス格納用変数 */

void main(void)
{
      unsigned int akib, akis;

      get_vec();      /* ベクタ取得関数 */
    /* バージョン取得ファンクション呼び出し */
      asm {
            mov ah,8
            call dword ptr [calladd]
            mov akib,ax
            mov akis,dx
      }
      printf("最大空きメモリブロックサイズ=%u\n", akib);
      printf("空きメモリサイズ=%u\n", akis);
}
```

10 EMBの割り当て /XMS

入力 **AH←09H**

DX←割り当てるEMBのサイズ（1kバイト単位）

出力 **AX**→1：割り当て成功

0：割り当て失敗

DX→割り当てメモリブロックのハンドル

解説 DXレジスタで指定した量のメモリをEMBに割り当てます．1Kバイト単位で指定することに注意してください．たとえば，DXレジスタの内容が1であれば，1KバイトのEMBが割り当てられます．

サンプル EMBを1KB割り当てます．

```
/*  このプログラムを実行するには，XMSドライバーが組み込まれている必要があります
   また，各サンプルに共通のget_vec()関数(XMS項目の最後に掲載)が必要です． */
#include <stdio.h>
#include <dos.h>

void get_vec(void);

unsigned long calladd; /* コールアドレス格納用変数 */

void main(void)
{
      unsigned int status, ha;

      get_vec();      /* ベクタ取得関数 */
    /* バージョン取得ファンクション呼び出し */
      asm {
            mov ah,9
```

```
			mov  dx,1
			call  dword ptr  [calladd]
			mov  status,ax
			mov  ha,dx
		}
		if(status == 1)  {
			printf("割り当てが成功しました\n");
			printf("ハンドル=%u\n",  ha);
		} else {
			printf("割り当てが失敗しました\n");
		}
}
```

11 EMBの開放　/XMS

入力　AH←0AH

DX←開放するEMBハンドル

出力　AX→1：開放成功

0：開放失敗

解説　DXレジスタで指定したハンドルのEMBを開放します．

サンプル　haに開放したいハンドルを指定して，そのハンドルのEMBを開放します

```
/*  このプログラムを実行するには，XMSドライバーが組み込まれている必要があります
   また，各サンプルに共通のget_vec()関数(XMS項目の最後に掲載)が必要です． */
#include  <stdio.h>
#include  <dos.h>

void  get_vec(void);

unsigned  long  calladd;

void  main(void)
{
	unsigned  int  ha;

	get_vec();
	ha  =  ????;	/*  ????にはハンドルの値を指定します  */
	asm  {
		mov  ah,0ah
		mov  dx,ha
		call  dword  ptr  [calladd]
		mov  status,ax
	}
	if(status == 1)  {
		printf("開放成功\n");
	} else {
		printf("開放失敗\n");
	}
}
```

12 EMBブロック転送 /XMS

入力　AH←0BH

DS：SI←転送のためのデータを格納したアドレス

出力　AX→1：転送成功

0：転送失敗

解説　DS：SIで示したアドレスに書き込まれているデータを元に，EMBのブロック転送を行います．DS：SIで示したアドレスには，次のようなデータを格納しておきます．

オフセット	内容
0000H	転送するデータの長さ　（4バイト）
0004H	転送元のハンドル　（2バイト）
0006H	転送元のオフセット　（4バイト）
000AH	転送先のハンドル　（2バイト）
000CH	転送先のオフセット　（4バイト）
0010H	

ここで，ハンドルは，ファンクション09Hで得たハンドル値を使用します．なお，コンベンショナルメモリが対象の場合は，ハンドル値に0を指定します．

また，コンベンショナルメモリの場合は，オフセットに，そのまま32ビットのオフセットを指定するのではなく，セグメント：オフセットといった指定をしなくてはいけません．

サンプル　テキストVRAMの内容をEMBに転送します．

```
/*  このプログラムを実行するには，XMSドライバーが組み込まれている必要があります
   また，各サンプルに共通のget_vec()関数(XMS項目の最後に掲載)が必要です． */
#include <stdio.h>
#include <dos.h>

struct xmstrans {    /* 転送情報を格納する構造体 */
        unsigned long length;
        unsigned int moto_ha;
        unsigned long moto_off;
        unsigned int saki_ha;
        unsigned long saki_off;
};

void main(void)
{
        unsigned int status, dataseg, dataoff;
```

```
    struct  xmstrans  data;

    get_vec();
    data.length  =  4000;
    data.moto_ha  =  0;                          /*  コンベンショナルメモリ指定  */
    data.moto_off  =  0xa0000000L;  /*  TVRAMアドレス  */
    data.saki_ha  =  ????;    /*  ????には転送先のEMBハンドルを指定します  */
    data.saki_off  =  0;                         /*  EMBオフセット=0  */
    dataseg  =  FP_SEG(&data);    /*  dataのセグメントとオフセットを得る  */
    dataoff  =  FP_OFF(&data);
/*  転送ファンクション呼び出し  */
    asm  {
          mov  ah,0bh
          mov  ds,dataseg
          mov  si,dataoff
          call  dword  ptr  [calladd]
          mov  status,ax
    }
    if(status  ==  1)  {
          printf("転送成功\n");
    }  else  {
          printf("転送失敗\n");
    }
}
```

13 EMBのロック/XMS

入力　AH←0CH
　　　DX←ロックするEMBハンドル

出力　AX→1：ロック成功
　　　　　0：ロック失敗

解説　DXレジスタで指定したEMBハンドルをロックします．

14 EMBのロック解除/XMS

入力　AH←0DH
　　　DX←ロックを解除するEMBハンドル

出力　AX→1：ロック解除成功
　　　　　0：ロック解除失敗

解説　DXレジスタで指定したEMBハンドルのロックを解除します．

15 EMBハンドル情報の取得 /XMS

入力 **AH←0EH**

DX←EMBハンドル

出力 **AX**→1：情報取得成功

0：情報取得失敗

BH→EMBのロックカウント

BL→EMBの空きハンドル数

DX→EMBのサイズ

解説 EMBハンドルの情報を取得します．

サンプル EMBハンドルの情報を取得し，表示します．

```
/*  このプログラムを実行するには，XMSドライバーが組み込まれている必要があります
    また，各サンプルに共通のget_vec()関数(XMS項目の最後に掲載)が必要です．*/
#include <stdio.h>
#include <dos.h>

void get_vec(void);

unsigned long calladd;

void main(void)
{
        unsigned int ha, size, status;
        unsigned char elock, aha;

        get_vec();
        ha = ????;      /* ????にはハンドルの値を指定します */
        asm {
                mov ah,0eh
                mov dx,ha
                call dword ptr [calladd]
                mov status,ax
                mov size,dx
                mov elock,bh
                mov aha,bl
        }
        if(status == 1) {
                printf("EMBのロックカウント=%d\n", elock);
                printf("EMBの空きハンドル数=%d\n", aha);
                printf("EMBのサイズ=%d\n", size);
        } else {
                printf("取得失敗\n");
        }
}
```

16 EMBの再割り当て /XMS

入力 AH←0FH

BX←再割り当てするEMBのサイズ

DX←再割り当てするEMBのハンドル

出力 AX→1：再割り当て成功

0：再割り当て失敗

解説 指定したEMBを再割り当てします．

17 UMBの割り当て /XMS

入力 AH←10H

DX←割り当てるUMBのサイズ（パラグラフ単位）

出力 AX→1：割り当て成功

0：割り当て失敗

BX→UMBのセグメント

解説 DXレジスタで指定されたサイズのUMBを割り当てます．

サンプル UMBを1パラグラフ割り当てます

```
/*  このプログラムを実行するには，XMSドライバーが組み込まれている必要があります
   また，各サンプルに共通のget_vec()関数(XMS項目の最後に掲載)が必要です． */
#include <stdio.h>
#include <dos.h>

void get_vec(void);

unsigned long calladd;

void main(void)
{
        unsigned int status, umbseg;

        get_vec();
        asm {
                mov ah,10h
                mov dx,1
                call dword ptr [calladd]
                mov status,ax
                mov umbseg,bx
        }
        if(status == 1) {
```

```
            printf("UMBの割り当てが成功しました\n");
            printf("UMBのセグメント=%04x\n", umbseg);
        } else {
            printf("UMBの割り当てが失敗しました\n");
        }
}
```

18 UMBの開放 /XMS

入力 **AH←11H**

DX←UMBのセグメント

出力 **AX→**1：開放成功

0：開放失敗

解説 UMBを開放します．

サンプル

```
/*  このプログラムを実行するには，XMSドライバーが組み込まれている必要があります
    また，各サンプルに共通のget_vec()関数(XMS項目の最後に掲載)が必要です． */
#include <stdio.h>
#include <dos.h>

void get_vec(void);

unsigned long calladd;

void main(void)
{
        unsigned int status, umbseg;

        umbseg = ????;    /* ????にはUMBのセグメントを設定してください */
        get_vec();
        asm {
                mov ah,11h
                mov dx,umbseg
                call dword ptr [calladd]
                mov status,ax
        }
        if(status == 1) {
                printf("UMBの開放に成功しました\n");
        } else {
                printf("UMBの開放に失敗しました\n");
        }
}
```

■サンプルプログラム

```
/*  EMBメモリにTVRAMの内容を待避します.
   また,待避した内容を復活します.                    */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>

unsigned char xms_ex(void);
void get_vec(void);
unsigned int xms_get(void);
unsigned int xms_trans(int, int);
unsigned int xms_end(unsigned int);

unsigned long calladd;

/* データ転送用構造体定義 */
struct xmstrans {
      unsigned long length;
      unsigned int moto_ha;
      unsigned long moto_off;
      unsigned int saki_ha;
      unsigned long saki_off;
};

void main(void)
{
      unsigned char status;
      unsigned int ha;
      int i;

      printf("\x1b*");
      if(xms_ex() != 0x80) {          /*  XMSドライバの存在確認  */
            printf("xmsドライバが存在しません\n");
            exit(-1);
      }
      get_vec();                           /*  XMSファンクションコールアドレス取得  */
      ha = xms_get();                      /*  EMBの割り当て  */

      for(i = 0; i < 20; i++) {       /*  テスト画面を書く  */
            printf("test test\n");
      }

      xms_trans(0, ha);                   /* TVRAM -> EMB ì]ëó */
      printf("何かキーを押すと,EMBにTVRAMの内容を待避します\n");
      getch();

      printf("\x1b*");
      printf("何かキーを押すと,EMBに待避した内容をTVRAMに復活します\n");
      getch();

      xms_trans(1, ha);                   /* EMB -> TVRAM 転送 */
```

```
	if(xms_end(ha) != 1) {		/* EMBの開放 */
		printf("EMBの開放に失敗しました\n");
	}
}

/* XMSドライバの存在チェック */
unsigned char xms_ex(void)
{
	unsigned char status;

	asm {
		mov ax,4300h
		int 2fh
		mov status,al
	}
	return status;
}

/* XMSファンクションコールアドレス取得 */
void get_vec(void)
{
	/* calladd という変数にコールアドレスを格納します */
	asm {
		mov ax,4310h
		int 2fh
		mov word ptr [calladd],bx
		mov word ptr [calladd+2],es
	}
}

/* EMBメモリを5KB割り当てます */
unsigned int xms_get(void)
{
	unsigned int status, ha;

	/* EMB割り当てファンクション呼び出し */
	asm {
		mov ah,9
		mov dx,5
		call dword ptr [calladd]
		mov status,ax
		mov ha,dx
	}
	if(status != 1) {
		printf("EMBの割り当てに失敗しました\n");
		exit(-1);
	} else {
		printf("EMBを5KB割り当てました\n");
	}
	return ha;
}

/* TVRAM<->EMB間の転送を行います */
/* sw = 0: TVRAM -> EMB
```

```
    sw = 1: EMB -> TVRAM */
unsigned int xms_trans(int sw, int ha)
{
        unsigned int status, dataseg, dataoff;
        struct xmstrans data;

        data.length = 4000;
        if(sw == 0) {
                data.moto_ha = 0;                 /* コンベンショナルメモリ指定 */
                data.moto_off = 0xa0000000L; /* TVRAMアドレス */
                data.saki_ha = ha;                /* EMBハンドル指定 */
                data.saki_off = 0;                /* EMBオフセット=0 */
        } else {
                data.moto_ha = ha;                /* EMBハンドル指定 */
                data.moto_off = 0;                /* EMBオフセット=0 */
                data.saki_ha = 0;                 /* コンベンショナルメモリ指定 */
                data.saki_off = 0xa0000000L; /* TVRAMアドレス */
        }
        dataseg = FP_SEG(&data); /* dataのセグメントとオフセットを得る */
        dataoff = FP_OFF(&data);
        /* 転送ファンクション呼び出し */
        asm {
                mov ah,0bh
                mov ds,dataseg
                mov si,dataoff
                call dword ptr [calladd]
                mov status,ax
        }
        return status;
}

/* ハンドルがhaのEMBメモリを開放します */
unsigned int xms_end(unsigned int ha)
{
        unsigned int status;

        /* EMB開放ファンクション呼び出しÅ@*/
        asm {
                mov dx,ha
                mov ah,0ah
                call dword ptr [calladd]
                mov status,ax
        }
        return status;
}
```

§2-9 ディスク

ディスクというのはコンピュータの記憶装置の1つで，記憶媒体を円盤状（ディスク状）にしたものです．

ディスクは，外側から内側に向かって，同心円状に分割されています（図2-34 ディスクの構造を参照）．分割されたそれぞれを，「トラック」と呼びます．また，トラックをある一定の間隔で区切ったものを「セクタ」といいます．ディスク装置のヘッドは，半径方向に対して動くことができるので，ヘッドはどんなトラックにも移動することができます．

中心から等距離にあるトラックのすべてを「シリンダ」といいます．すなわち，両面ディスクの場合は，1シリンダは2トラックで構成されているということになります．

そして，ディスク装置はセクタ単位で読み込み／書き込みが行われます．ディスクの中の特定の場所の処理をするときは，そのヘッド番号H，シリンダ番号C，セクタ番号R，を指定すればその場所は一意に定まります．

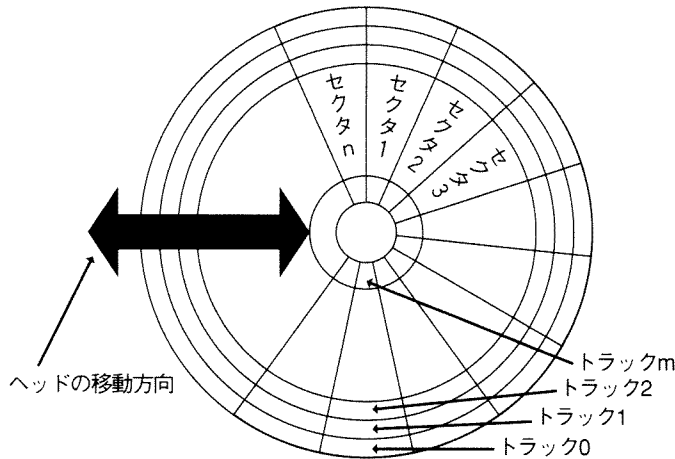



図2-34 ディスクの構造

よって，ある指定したセクタの処理をするときは，ヘッドをそのトラックの場所まで移動して，ディスクを回転させれば，ヘッドのところに指定したセクタの場所がきて，その処理ができるとことになります．

フロッピーディスクはフロッピーディスク媒体をディスク装置から着脱可能に対し，ハードディスクは媒体が装置の中に固定されており，着脱が不能という点を除けば，おおよそその原理的な仕組み，構造は同じものです．

■2-9-1 ――― フロッピーディスク

●98のフロッピーインターフェースとμPD765A相当

98には，フロッピーディスクのインターフェースとして，1.44MB，1MBフロッピーディスク，640KBフロッピーディスクがあります．これらのインターフェースは，フロッピーディスクコントローラ（FDC）用のLSI・μPD765A相当を中心にして構成されています．なお，μPD765A相当には4台までフロッピー装置をつなぐことができます．FDCとメモリ間でのデータ転送は，DMAによって行われます．DMAは，DMAコントローラμPD8237相当によって制御されます．現在では，ほとんどの98が1MB／640KBインターフェースを標準装備していますので，それについて少し説明をします．

1MB／640KBインターフェースは，内蔵の2台のフロッピードライブと外付けの2台の1MBのドライブを制御できます．内蔵のフロッピードライブに対しては，1MB／640KB両用インタフェースとして動作しますが，外付けのフロッピードライブに対しては，1MB専用としてしか作動かしません．1MB／640KBインターフェースには，1MBモードと640KBモードの2つのモードがあります．ノーマルモードでは，2つのモードは独立したI／Oアドレスを持っており，モード切り替えはI／Oアドレスとモード切り替え用のI／Oアドレスによって行われます．ハイレゾモードでは2つのモードのI／Oアドレスは独立していません．そのために別にモード切り替え用のI／Oアドレスがあり，それによってモードを切り替えます．そのほか，ノーマルモードとハイレゾモードでは割り込みレベルやDMAチャネルなどが違います．

このように，ディスク装置は，ディスクドライブ，FDC，DMACなどにより構成されています．そして，FDC，DMAにはI／Oポートが定まっていますので，直接I／Oポートにデータを書き込んで入出力の命令を実行をすればいいのですが，それぞれのハードウェアに対して高度な知識が必要とされ，また複雑な処理を行いますので，結果としてディスク装置の入出力制御を非常に煩わしく，煩雑にさせてしまいます．そこで，ディスク装置を安全に，また簡単に入出力制御ができるようにディスクBIOSが用意されています．

■2-9-2 ――― ハードディスク

先ほど述べたように，基本的な構造はフロッピーディスクとほとんど変わりません．ハードディスクは，フロッピーディスクと同じ様にヘッド番号，シリンダ番号，セクタ番号を指定すれば，アクセスができます．このようにヘッド番号，シリンダ番号，セクタ番号で指定される場所を「絶対アドレス」といいます．これに対して，「相対アドレス」でハードディスクをアクセスすることもできます．「絶対アドレス」と「相対アドレス」の関係は次のセクションのハードディスクBIOSの一般形式を参照してください．

絶対アドレスによるアクセスと，相対アドレスによるアクセスは，デバイスタイプ識別コードで使い分けることができます（表2-34）．

表2-34 DA／UAと対応するディスクドライブ

DA／UA	対応するディスクドライブ
00H～03H	SASIハードディスク（相対アドレス指定）
10H～13H	両用インターフェースの1MBモードのときの，640KBフォーマットのディスクアクセスモード
20H～27H	SCSIハードディスク（相対アドレス指定）
30H～33H	1.44MB対応両用インタフィースのアクセスモード
4xH	未使用
50H～53H	320KBフロッピーディスク
6xH	未使用
70H～73H	640KBインターフェース，両用インターフェースの640KBモード時の，640KBフォーマットのディスクアクセスモード
80H～83H	SASIハードディスク（絶対アドレス指定）
90H～93H	1MBインターフェース，両用インターフェースの1MBモード時の，1MBフォーマットのディスクアクセスモード
A0H～A7H	SCSIハードディスク（絶対アドレス指定）
B0H～B3H	1.44MB対応両用インターフェースのアクセスモード
C0H～C7H	SCSIデバイス
DxH	未使用
ExH	未使用
F0H～F3H	両用インターフェースの640KBモード時の，1MBフォーマットのフロッピーアクセスモード

●フロッピーディスク BIOSコマンドの一般形式

割り込み INT 1BH

DISK BIOSのソフトウェア割り込みはINT 1BHをつかいます.

入　力

AH＝BIOSコマンド識別コード
AL＝デバイスアドレスコード（DA：Device Address）
／ユニット番号（UA：Unit Address）
BX＝転送データ長（バイト単位）
CH＝セクタ長
CL＝シリンダ番号
DH＝ヘッド番号
DL＝セクタ番号
ES：BP＝データバッファ領域の先頭アドレス

◆AH＝BIOSコマンド識別コード

レジスタAHの下位の4ビット（0～3ビット）でコマンド識別コードを指定します．上位の4ビット（4～7ビット）でそのコマンドによって，SEEK動作の選択，リトライ動作の選択，単密度（FM）／倍密度（MFM）の読み出しの選択，シングルトラック／マルチトラックの選択が可能です．コマンドに該当しないものを指定したときには，正常終了をします．

◆AL＝デバイスタイプ識別コード（DA：Device Address）
／ユニット番号（UA：Unit Address）

UAは，フロッピーディスク装置の場合下位2ビットがドライブの番号に当たります．つまり，1台めなら00，2台めなら01，……，4台めなら11，といった具合です．DA／UAと選択されるデバイスとの関係は表2-34を参照してください．

出　力

CF＝終了条件（0：正常終了／1：異常終了）
AH＝ステータス情報（ステータス情報一覧表を参照）

●フロッピーディスク BIOSコマンド使用上の注意

・データバッファは，複数のDMAバンクにまたがってはいけません．
（286以上のマシンでは，BIOSを使って制御する場合には，関係ありません．）

・ライト時のデータバッファの大きさは，物理セクタ長の整数倍としてください．

■フロッピーディスクBIOS一覧表（INT 1BH）

機能コード	機能	1MBFD	640KBFD	1M/640KB両用FD		1.44MB対応両用FD	
				1MBFD	640KBFD	1.44MBFD	640KBFD
01H	ベリファイ	○	○	○	○	○	○
02H	診断のための読み出し	○	×	○	○	○	○
03H	初期化	○	○	○	○	○	○
04H	センス	○	○	○	○	○	○
05H	データの書き込み	○	○	○	○	○	○
06H	データの読み出し	○	○	○	○	○	○
07H	シリンダ0へのシーク	○	○	○	○	○	○
09H	デリーデッドデーの書き込み	○	×	○	×	○	×
0AH	IDの読み出し	○	○	○	○	○	○
0CH	デリーデッドデーの読み出し	○	×	○	×	○	×
0DH	トラックのフォーマット	○	○	○	○	○	○
0EH	動作モードの設定	×	×	○	×	○	×
10H	シーク	○	○	○	○	○	○
83H	モータ停止モードの設定	×	×	○	○	○	○
83H	初期化	×	×	×	○	×	○

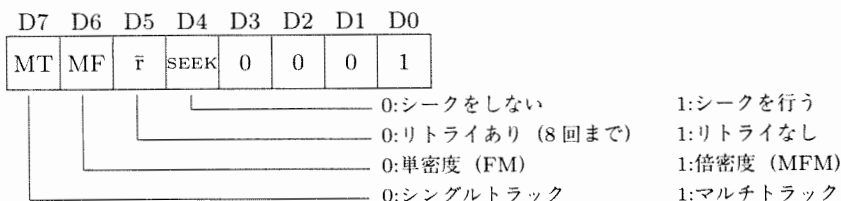
■フロッピーディスクBIOSステータス一覧表

AH	CF	説明
00H	0	正常終了
10H	0／1	DDAMを検出した
20H	1	バッファ領域のアドレスがDMAバンクにまたがっている
30H	1	1回でできるデータの転送容量を超えたデータ長を指定した
40H	1	デバイスからFault信号を受け取った 一定時間内にシリンダ0にシークできなかった DA／UAが不適当
50H	1	一定時間内に，セクタ・メモリ間のデータの転送が終了出来なかった
60H	1	ディスクドライブがReady状態でない
70H	1	Write Protect信号がオンの状態
80H	1	そのほかのエラー
90H	1	FDCのアクセスのとき，一定時間内に処理が終わらなかった
A0H	1	IDを読みだしたときに，CRCエラーが発生した
B0H	1	データを読読みだしたときに，CRCエラーが発生した
C0H	1	指定したセクタが，トラック上になかった
D0H	1	指定したシリンダが見つからなかった
E0H	1	IDが見つからなかった ID検出後，DAMを検出できなかった
F0H	1	データを読み込むときにDAMまたはDDAMを検出できなかった

ベリファイ (VERIFY) FDD

機能コード 01H

割り込み INT 1BH

入　　力 AH＝BIOS コマンド識別コード: (0H～FH) × 10H + 01H

D7	D6	D5	D4	D3	D2	D1	D0
MT	MF	$\bar{r}$	SEEK	0	0	0	1

- D4 (SEEK): 0:シークをしない　1:シークを行う
- D5 ($\bar{r}$): 0:リトライあり（8回まで）　1:リトライなし
- D6 (MF): 0:単密度（FM）　1:倍密度（MFM）
- D7 (MT): 0:シングルトラック　1:マルチトラック

AL＝デバイスアドレス番号（DA／UA）(p.264)
BX＝転送データ長（バイト単位）
CH＝セクタ長
CL＝開始シリンダ番号
DH＝ヘッド番号（0・1）
DL＝開始セクタ番号
ES:BP＝データバッファ領域の先頭アドレス

出　　力 CF＝終了条件　0:正常終了　1:異常終了
AH＝ステータス情報（p.267:ステータス情報一覧表を参照）

機　　能 指定されたデバイスアドレス番号，シリンダ番号，ヘッド番号に対応するトラックの開始セクタから，指定したデータ長のデータを読み取ります．読み取ったデータは，メモリに格納されません．読み取ったデータをメモリに格納しないことを除けば，そのほかは「データの読みだし（機能コード 06H）」と同じです．なお，DDAM (Deleted Data Address Mark) を検出したときは，そのセクタをスキップして，処理を実行します．

サンプル データのベリファイをするプログラムです．1MB インターフェース，ユニット番号 0 のデバイスの，シリンダ番号 10，ヘッド番号 0，開始セクタを 5 として 2 セクタ分だけデータのベリファイをします．

```
#include <stdio.h>
#include <dos.h>

#define AH            0x71                /* 01110001          */
#define BYTE_SECTOR   1024                /* 1024バイト／セクタ  */
#define DA_UA         0x90                /* 1MB, ユニット番号0  */
#define BYTE          BYTE_SECTOR * 2     /* 1024*2 バイト読みだす */
#define CYLINDER      10                  /* シリンダ番号10      */
#define HEAD          0                   /* ヘッド番号    0    */
```

```
#define SECTOR_START     5                  /* 開始セクタ    5          */
#define SECTOR_N         3                  /* セクタ長      3          */

void main(void)
{
    struct REGPACK regs;

    unsigned char buf[BYTE];     /* 2セクタ分のバッファを確保 */
    int i;

    for (i = 0;i < BYTE;i ++)     /* バッファをクリア */
        buf[i] = 0;

    regs.r_ax = AH<<8 | DA_UA;
    regs.r_bx = BYTE;
    regs.r_cx = SECTOR_N<<8 | CYLINDER;
    regs.r_dx = HEAD<<8 | SECTOR_START;
    regs.r_es = FP_SEG(buf);     /* バッファのセグメントを得る */
    regs.r_bp = FP_OFF(buf);     /* バッファのオフセットを得る */

    intr(0x1b,&regs);

    if (regs.r_flags & 1 == 1){
        printf("異常終了です¥n");
    }

    else{
        for(i = 0;i < 2;i ++)
            printf("sector = %d [0] = %x [1023] = %x¥n",
                SECTOR_START + i , buf[i * BYTE_SECTOR],
                buf[(i + 1) * BYTE_SECTOR - 1]);

        printf("正常に終了しました¥n");
    }
}
```

2 診断のためのデータの読みだし (READ DIAGNOSTIC) FDD

機能コード 02H

割り込み INT 1BH

入　　力 AH＝BIOS コマンド識別コード: (0H～7H) × 10H + 02H

D7	D6	D5	D4	D3	D2	D1	D0
0	MF	$\bar{r}$	SEEK	0	0	1	0

- SEEK — 0: シークをしない　1: シークを行う
- $\bar{r}$ — 0: リトライあり (8回まで)　1: リトライなし
- MF — 0: 単密度 (FM)　1: 倍密度 (MFM)

AL＝デバイスアドレス番号（DA／UA）(p.264)
BX＝転送データ長（バイト単位）
CH＝セクタ長
CL＝開始シリンダ番号
DH＝ヘッド番号（0・1）
DL＝開始セクタ番号
ES:BP＝データバッファ領域の先頭アドレス

出　　力 CF＝終了条件　0：正常終了　1：異常終了
AH＝ステータス情報（p.267:ステータス情報一覧表を参照）

機　　能 指定されたセクタから順に指定されたデータ長だけ読み取り，データバッファ領域の先頭アドレスから格納します．ID，およびデータ部で読み取りエラーが起きても，読み取りを続けます．また，DAM，DDAM に対して影響されません．そのことを除けば，「データの読みだし」と同じ機能をします．データは，物理セクタ順に指定されたデータ数だけ読み取っていきます．

サンプル 診断のための読みだしをするプログラムです．1MB インターフェース，ユニット番号 0 のデバイスで，シリンダ番号 10，ヘッド番号 0，開始セクタを 5 として 2 セクタ分だけデータを読み出します

```
#include <stdio.h>
#include <dos.h>

#define AH              0x72                /* 01110010              */
#define BYTE_SECTOR     1024                /* 1024バイト／セクタ     */
#define DA_UA           0x90                /* 1MB, ユニット番号0     */
#define BYTE            BYTE_SECTOR * 2     /* 1024*2 バイト読みだす */
#define CYLINDER        10                  /* シリンダ番号10         */
#define HEAD            0                   /* ヘッド番号     0       */
#define SECTOR_START    5                   /* 開始セクタ     5       */
#define SECTOR_N        3                   /* セクタ長       3       */

void main(void)
{
    struct REGPACK regs;

    unsigned char buf[BYTE];     /* 2セクタ分のバッファを確保 */
    int i;

    for (i = 0;i < BYTE;i ++)     /* バッファをクリア */
        buf[i] = 0;

    regs.r_ax = AH<<8 | DA_UA;
    regs.r_bx = BYTE;
    regs.r_cx = SECTOR_N<<8 | CYLINDER;
    regs.r_dx = HEAD<<8 | SECTOR_START;
```

```
    regs.r_es = FP_SEG(buf);     /* バッファのセグメントを得る */
    regs.r_bp = FP_OFF(buf);     /* バッファのオフセットを得る */

    intr(0x1b,&regs);

    if (regs.r_flags & 1 == 1){
        printf("異常終了です¥n");
    }

    else{
        for(i = 0;i < 2;i ++)
            printf("sector = %d [0] = %x [1023] = %x¥n",
                SECTOR_START + i , buf[i * BYTE_SECTOR],
                buf[(i + 1) * BYTE_SECTOR - 1]);

        printf("正常に終了しました¥n");
    }
}
```

3 初期化 (INITIALIZE) FDD

機能コード 03H

割り込み INT 1BH

入　　力 AH=BIOS コマンド識別コード: 03H

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	1	1

AL=デバイスアドレス番号 (p.264)

出　　力 CF=終了条件　0：正常修了　1：異常終了

AH=ステータス情報 (p.267)

機　　能 フロッピーディスク装置全体の初期化を行います．このときシリンダ0へのシークも行います．

サンプル フロッピーディスク装置を初期化します．

```
#include <stdio.h>
#include <dos.h>

#define    AH       0x03        /*  00000011            */
#define    DA_UA    0x90        /*  1MB・ユニット番号0    */

void main(void)
{
    union REGS inregs,outregs;
```

```
        inregs.h.ah = AH;
        inregs.h.al = DA_UA;
            int86(0x1b,&inregs,&outregs);
        if (outregs.x.flags & 1 == 1){
            printf("エラーです¥n");
        }
        else{
            printf("初期化しました¥n");
        }

    }
```

4 初期化（INITIALIZE） FDD

（640KB／1MB両用インターフェースの，640KBモード専用）

機能コード 83H

割り込み INT 1BH

入　　力 AH＝BIOSコマンド識別コード：83H

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	1	1

AL＝デバイスアドレス番号（p.264）

出　　力 CF＝終了条件　0：正常終了　1：異常終了

AH＝ステータス情報（p.267）

機　　能 フロッピーディスク装置全体を，アテンショインインタラプトAI（FDDの状態遷移があったときに発せられる割り込み）を検出するように初期化を行います．AIなしに戻すことはできません．この時，シリンダ0へのシークも行います。

5 センス(SENSE) FDD

機能コード 04H

割り込み INT 1BH

入　　力 AH＝BIOSコマンド識別コード：04H または 24H

D7	D6	D5	D4	D3	D2	D1	D0
0	0	r	0	0	1	0	0

r — 0: リトライあり（8回まで）　1: リトライなし

AL＝デバイスアドレス番号（p.264）

出　　力	CF＝終了条件　0：正常終了　1：異常終了 AH＝ステータス情報（p.267:ステータス情報一覧表を参照）
機　　能	指定したデバイスアドレス番号の装置の状態を調べます.
サンプル	機能コード 84H のサンプルを参照してください.

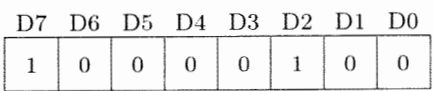
6 センス(640KB／1MB両用インターフェース専用) (SENSE) FDD

（640KB ／ 1MB 両用インターフェース専用）

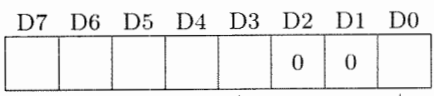
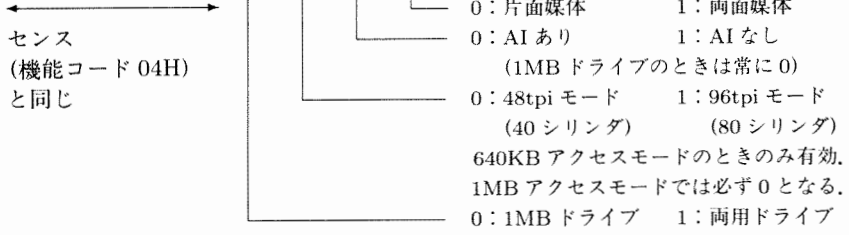
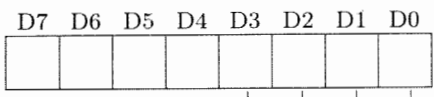
機能コード 84H

割り込み INT 1BH

入　　力 AH＝BIOS コマンド識別コード：84H

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	1	0	0

AL＝デバイスアドレス番号 (p.264)

出　　力 CF＝終了条件　0：正常終了　1：異常終了
AH＝ステータス情報（p.267:ステータス情報一覧表を参照）

● 1MB インターフェースの時

● 640KB インターフェースの時

機　　能 指定されたデバイスアドレス番号の状態を調べます．

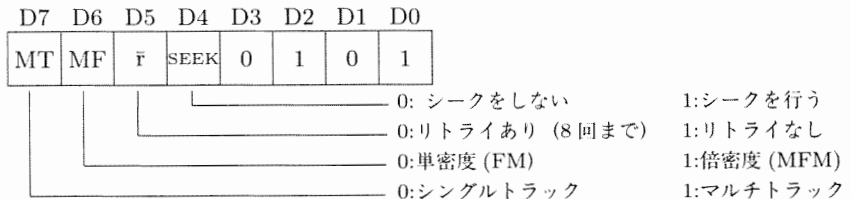
サンプル 1MBインターフェース，ユニット番号0のデバイスでセンスコマンドを実行します

```
#include <stdio.h>
#include <dos.h>

#define    AH    0x84        /*  10000100            */
#define    DA_UA    0x90        /*  1MB・ユニット番号0    */

void main(void)
{
    union REGS inregs,outregs;
    inregs.h.ah = AH;
    inregs.h.al = DA_UA;
    int86(0x1b,&inregs,&outregs);

    printf("AHの内容は %2x です. ¥n",outregs.h.ah);
}
```


7 データの書き込み（WRITE DATA） FDD

機能コード 05H

割り込み INT 1BH

入　　力 AH＝BIOSコマンド識別コード：（0H～FH）× 10H ＋ 05H

D7	D6	D5	D4	D3	D2	D1	D0
MT	MF	r̄	SEEK	0	1	0	1

- SEEK — 0: シークをしない　1:シークを行う
- r̄ — 0:リトライあり（8回まで）　1:リトライなし
- MF — 0:単密度 (FM)　1:倍密度 (MFM)
- MT — 0:シングルトラック　1:マルチトラック

AL＝デバイスアドレス番号（DA／UA）(p.264)
BX＝転送データ長（バイト単位）
CH＝セクタ長
CL＝開始シリンダ番号
DH＝ヘッド番号（0・1）
DL＝開始セクタ番号
ES：BP＝データバッファ領域の先頭アドレス

出　　力 CF＝終了条件　　0：正常終了　　1：異常終了
AH＝ステータス情報（p.267:ステータス情報一覧表を参照）

機　能 指定されたデータバッファ領域のデータを，指定されたデータの長さ（データ長）分だけ，指定されたデバイスアドレス番号，シリンダ番号，ヘッド番号に対応するトラックの開始セクタから書き込みます．転送はDMAにより行われますので，データバッファ領域が複数のDMAバンクにまたがってはいけません（286以上のマシンでは，BIOSを使って制御するだけなら，関係ありません）．このデータ書き込みでは，マルチトラック書き出しの指定はできません．マルチトラックの指定をすると，μPD765は書き込み時には正常に作動かしないのでマルチトラックの指定はしないでください．なお，データの書き込みがセクタの途中で終了したときは，そのセクタの残りの部分には00Hが書き込まれます．

サンプル ディスクデータの書き込みをするプログラムです．1MBインターフェース，ユニット番号0のデバイスに，シリンダ番号10，ヘッド番号0，開始セクタを5として2セクタ分だけデータ10Hを書き込みます．

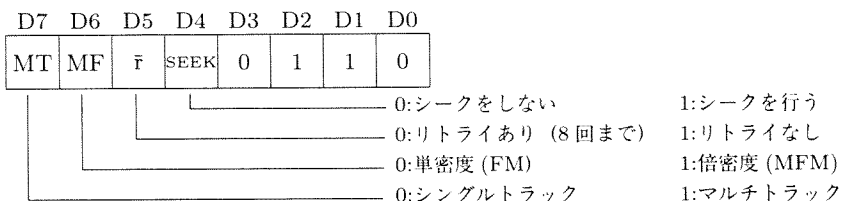
```
#include <stdio.h>
#include <dos.h>

#define AH              0x75               /* 01110101            */
#define BYTE_SECTOR     1024               /* 1024バイト／セクタ   */
#define DA_UA           0x90               /* 1MB，ユニット番号0   */
#define BYTE            BYTE_SECTOR * 2    /* 1024*2 バイト読みだす */
#define CYLINDER        10                 /* シリンダ番号10       */
#define HEAD            0                  /* ヘッド番号     0     */
#define SECTOR_START    5                  /* 開始セクタ     5     */
#define SECTOR_N        3                  /* セクタ長       3     */

void main(void)
{
    struct REGPACK regs;

    unsigned char buf[BYTE];     /* 2セクタ分のバッファを確保 */
    int i;

    for (i = 0;i < BYTE;i ++)     /* バッファのセット */
        buf[i] = 0x10;

    regs.r_ax = AH<<8 | DA_UA;
    regs.r_bx = BYTE;
    regs.r_cx = SECTOR_N<<8 | CYLINDER;
    regs.r_dx = HEAD<<8 | SECTOR_START;
    regs.r_es = FP_SEG(buf);     /* バッファのセグメントを得る */
    regs.r_bp = FP_OFF(buf);     /* バッファのオフセットを得る */

    intr(0x1b,&regs);

    if (regs.r_flags & 1 == 1){
        printf("異常終了です¥n");
    }
```

```
        else{
            printf("正常に終了しました¥n");

        }
    }
```

7' データの読みだし（READ DATA） FDD

機能コード 06H

割り込み INT 1BH

入　　力 AH＝BIOSコマンド識別コード：(0H〜FH) × 10H ＋ 06H

AL＝デバイスアドレス番号（DA／UA）(p.264)
BX＝転送データ長（バイト単位）
CH＝セクタ長
CL＝開始シリンダ番号
DH＝ヘッド番号（0・1）
DL＝開始セクタ番号
ES:BP＝データバッファ領域の先頭アドレス

出　　力 CF＝終了条件　0:正常終了　1:異常終了
AH＝ステータス情報（ステータス情報一覧表を参照）

機　　能 指定されたデバイスアドレス番号，シリンダ番号，ヘッド番号に対応するトラックの開始セクタから，指定されたデータ長のデータをデータバッファ領域の先頭アドレスに読み出します．また，データバッファへの転送はDMAを介して行われますので，データバッファの領域は複数のDMAバンクにまたがってはいけません（286以上のマシンでは，BIOSを使って制御するだけなら，関係ありません）．データの読みだしは，指定されたシリンダ内に限られます．さらに，シングルトラックの指定をした時は，指定されたヘッドに限られます．マルチトラックを指定したときは，指定された読みだし開始のヘッド番号が0であれば，同じシリンダ内のヘッド番号1で読み出せるデータも読みだすことが出来ます．

サンプル ディスクデータの読みだしをするプログラムです．シリンダ番号10，ヘッド番号0，開始セクタを5として2セクタ分だけデータをよみだします．

```
#include <stdio.h>
#include <dos.h>

#define AH           0x76                /* 01110110              */
#define BYTE_SECTOR  1024                /* １０２４バイト／セクタ  */
#define DA_UA        0x90                /* １ＭＢ，ユニット番号０  */
#define BYTE         BYTE_SECTOR * 2     /* 1024*2 バイト読みだす   */
#define CYLINDER     10                  /* シリンダ番号１０        */
#define HEAD         0                   /* ヘッド番号    0         */
#define SECTOR_START 5                   /* 開始セクタ    5         */
#define SECTOR_N     3                   /* セクタ長      3         */

void main(void)
{
    struct REGPACK regs;

    unsigned char buf[BYTE];      /* ２セクタ分のバッファを確保 */
    int i;

    for (i = 0;i < BYTE;i ++)      /* バッファをクリア */
        buf[i] = 0;

    regs.r_ax = AH<<8 | DA_UA;
    regs.r_bx = BYTE;
    regs.r_cx = SECTOR_N<<8 | CYLINDER;
    regs.r_dx = HEAD<<8 | SECTOR_START;
    regs.r_es = FP_SEG(buf);     /* バッファのセグメントを得る */
    regs.r_bp = FP_OFF(buf);     /* バッファのオフセットを得る */

    intr(0x1b,&regs);

    if (regs.r_flags & 1 == 1){
        printf("異常終了です¥n");
    }

    else{
        for(i = 0;i < 2;i ++)
            printf("sector = %d [0] = %x [1023] = %x¥n",
                SECTOR_START + i , buf[i * BYTE_SECTOR],
                buf[(i + 1) * BYTE_SECTOR - 1]);

        printf("正常に終了しました¥n");
    }
}
```

8 シリンダ0へのシーク (RECALIBRATE) FDD

機能コード 07H

割り込み INT 1BH

入　　力 AH=BIOS コマンド識別コード: 07H または 27H

D7	D6	D5	D4	D3	D2	D1	D0
0	0	r̄	0	0	1	1	1

D5: 0: リトライあり (8回まで)　1: リトライなし

AL=デバイスアドレス番号 (p.264)

出　　力 CF=終了条件　0：正常終了　1：異常終了

AH=ステータス情報 (p.267:ステータス情報一覧表を参照)

機　　能 指定されたデバイスタイプユニット番号のヘッドを，シリンダ番号0までシークさせます．シーク動作は，シリンダ番号0の方向へ1シリンダずつ行い，装置からのトラック0信号を検出するまで繰り返します．

サンプル 1MB インターフェース，ユニット番号0のデバイスで，0シリンダへのシークコマンドを実行します．

```
#include <stdio.h>
#include <dos.h>

#define    AH       0x07       /* 00000111          */
#define    DA_UA    0X90       /* 1MB・ユニット番号0*/

void main(void)
{
    union REGS inregs,outregs;
    inregs.h.ah = AH;
    inregs.h.al = DA_UA;
    int86(0x1b,&inregs,&outregs);
    if (outregs.x.flags & 1 == 1){
        printf("エラーです¥n");
    }
    else{
        printf("正常に終了しました¥n");
    }

}
```

9 デリーテッドデータの書き込み (WRITE DATA) FDD

機能コード 09H

割り込み INT 1BH

入　　力 AH＝BIOS コマンド識別コード：(0H～FH) × 10H ＋ 09H

AL＝デバイスアドレス番号 (DA／UA) (p.264)
BX＝転送データ長 (バイト単位)
CH＝セクタ長
CL＝開始シリンダ番号
DH＝ヘッド番号 (0・1)
DL＝開始セクタ番号
ES：BP＝データバッファ領域の先頭アドレス

出　　力 CF＝終了条件　0：正常終了　1：異常終了
AH＝ステータス情報 (p.267:ステータス情報一覧表を参照)

機　　能 データの書き込みのときに，セクタのデータフィールドの DAM (Data Address Mark) のかわりに，DDAM (Deleated Data Address Mark) を書き込みます．そのほかの機能は，「データの書き込み　機能コード 05H」と同じです．

サンプル 1MB インターフェース，ユニット番号 0 のデバイスに，シリンダ番号 10，ヘッド番号 0，開始セクタを 5 として 2 セクタ分だけデータ 10H を書き込みます．

```
#include <stdio.h>
#include <dos.h>

#define AH              0x79            /* 01111001            */
#define BYTE_SECTOR     1024            /* 1024バイト／セクタ    */
#define DA_UA           0x90            /* 1MB, ユニット番号0    */
#define BYTE            BYTE_SECTOR * 2 /* 1024*2 バイト読みだす */
#define CYLINDER        10              /* シリンダ番号10        */
#define HEAD            0               /* ヘッド番号     0      */
C#define SECTOR_START   5               /* 開始セクタ     5      */
#define SECTOR_N        3               /* セクタ長       3      */

void main(void)
{
```

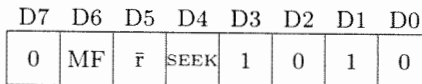
```
        struct REGPACK regs;

        unsigned char buf[BYTE];      /* 2セクタ分のバッファを確保 */
        int i;

        for (i = 0;i < BYTE;i ++)      /* バッファのセット */
            buf[i] = 0x10;

        regs.r_ax = AH<<8 | DA_UA;
        regs.r_bx = BYTE;
        regs.r_cx = SECTOR_N<<8 | CYLINDER;
        regs.r_dx = HEAD<<8 | SECTOR_START;
        regs.r_es = FP_SEG(buf);     /* バッファのセグメントを得る */
        regs.r_bp = FP_OFF(buf);     /* バッファのオフセットを得る */

        intr(0x1b,&regs);

        if (regs.r_flags & 1 == 1){
            printf("異常終了です¥n");
        }
        else{
            printf("正常に終了しました¥n");

        }
    }
```

10 IDの読みだし（READ ID） FDD

機能コード 0AH

割り込み INT 1BH

入　　力 AH＝BIOS コマンド識別コード：(0H〜7H) × 10H ＋ 0AH

D7	D6	D5	D4	D3	D2	D1	D0
0	MF	r̄	SEEK	1	0	1	0

- SEEK　0: シークをしない　1: シークを行う
- r̄　0: リトライあり（8回まで）　1: リトライなし
- MF　0: 単密度（FM）　1: 倍密度（MFM）

AL＝デバイスアドレス番号（DA／UA）(p.264)
CL＝開始シリンダ番号
DH＝ヘッド番号（0・1）

出　　力 CF＝終了条件　0：正常終了　1：異常終了
AH＝ステータス情報（p.267:ステータス情報一覧表を参照）
CH＝セクタ長

CL＝シリンダ番号
DH＝ヘッド番号
DL＝セクタ番号

機　　能　指定されたデバイアドレス番号の指定トラック上の正常に読み取れたIDをID情報として各レジスタに格納します.

サンプル　1MBインターフェースの0番ユニットのヘッド番号0, シリンダ番号0のIDを取得します.

```
#include <stdio.h>
#include <dos.h>

#define AH          0x5a    /*01011010                      */
#define DEVICE      0x90    /*1MBインターフェースモード・0番    */
#define CYLINDER    0       /*シリンダ番号0                  */
#define HEAD        1       /*ヘッド番号1                     */

void main(void)
{
    union REGS inregs,outregs;
    inregs.h.ah = AH;
    inregs.h.al = DEVICE;
    inregs.h.cl = CYLINDER;
    inregs.h.dh = HEAD;
    int86(0x1b,&inregs,&outregs);
    if (outregs.x.flags & 1 == 1){
        printf("エラーです¥n");
    }
    else{
        printf("セクタ長      %d¥n",outregs.h.ch);
        printf("シリンダ番号  %d¥n",outregs.h.cl);
        printf("ヘッド番号    %d¥n",outregs.h.dh);
        printf("セクタ番号    %d¥n",outregs.h.dl);
    }
}
```

11 デリーテッドデータの読みだし (READ DELETED DATA) FDD

機能コード 0CH

割り込み INT 1BH

入　力 AH＝BIOS コマンド識別コード: (0H〜FH) × 10H ＋ 0CH

D7	D6	D5	D4	D3	D2	D1	D0
MT	MF	$\bar{\text{r}}$	SEEK	1	1	0	0

- D4 (SEEK): 0: シークをしない　1: シークを行う
- D5 ($\bar{\text{r}}$): 0: リトライあり (8回まで)　1: リトライなし
- D6 (MF): 0: 単密度 (FM)　1: 倍密度 (MFM)
- D7 (MT): 0: シングルトラック　1: マルチトラック

AL＝デバイスアドレス番号 (DA／UA) (p.264)
BX＝転送データ長 (バイト単位)
CH＝セクタ長
CL＝開始シリンダ番号 (0〜76)
DH＝ヘッド番号 (0・1)
DL＝開始セクタ番号 (1〜26)
ES：BP＝データバッファ領域の先頭アドレス

出　力 CF＝終了条件　0：正常終了　1：異常終了
AH＝ステータス情報 (p.267:ステータス情報一覧表を参照)

機　能 セクタのデータフィールドに DDAM (Dleted Data Address Mark) があるものを読み取ります．DDAM を扱うという以外は,「データの読みだし　機能コード 06H」と同じ機能を持ちます．

サンプル デリーテッドデータの読みだしをするプログラムです．シリンダ番号 0，ヘッド番号 0，開始セクタを 1 として 2 セクタ分だけデリーテッドデータを読みだします

```
#include <stdio.h>
#include <dos.h>

#define AH           0x7c              /* 01111100          */
#define BYTE_SECTOR  1024              /* 1024バイト／セクタ   */
#define DA_UA        0x90              /* 1MB, ユニット番号0   */
#define BYTE         BYTE_SECTOR * 2   /* 1024*2 バイト読みだす */
#define CYLINDER     0                 /* シリンダ番号  0     */
#define HEAD         0                 /* ヘッド番号    0     */
#define SECTOR_START 1                 /* 開始セクタ    1     */
#define SECTOR_N     3                 /* セクタ長      3     */

void main(void)
{
    struct REGPACK regs;
```

```
    unsigned char buf[BYTE];      /* 2セクタ分のバッファを確保 */
    int i;

    for (i = 0;i < BYTE;i ++)      /* バッファをクリア */
        buf[i] = 0;

    regs.r_ax = AH<<8 | DA_UA;
    regs.r_bx = BYTE;
    regs.r_cx = SECTOR_N<<8 | CYLINDER;
    regs.r_dx = HEAD<<8 | SECTOR_START;
    regs.r_es = FP_SEG(buf);    /* バッファのセグメントを得る */
    regs.r_bp = FP_OFF(buf);    /* バッファのオフセットを得る */

    intr(0x1b,&regs);

    if (regs.r_flags & 1 == 1){
        printf("異常終了です¥n");
    }
    else{
        for(i = 0;i < 2;i ++)
            printf("sector = %d [0] = %x [1023] = %x¥n",
                SECTOR_START + i , buf[i * BYTE_SECTOR],
                buf[(i + 1) * BYTE_SECTOR - 1]);

        printf("正常に終了しました¥n");

    }
}
```

12 トラックのフォーマット FDD

機能コード 0DH

割り込み INT 1BH

入　　力 AH＝BIOS コマンド識別コード：(0H～7H) × 10H ＋ 0DH

D7	D6	D5	D4	D3	D2	D1	D0
0	MF	$\bar{\text{r}}$	SEEK	1	1	0	1

- SEEK — 0: シークをしない　1: シークを行う
- $\bar{\text{r}}$ — 0: リトライあり（8回まで）　1: リトライなし
- MF — 0: 単密度（FM）　1: 倍密度（MFM）

AL＝デバイスアドレス番号（DA／UA）(p.264)
BX＝転送データ長（バイト単位）
CH＝セクタ長
CL＝シリンダ番号
DH＝ヘッド番号（0・1）
DL＝データ部への書き込みデータパターン
ES：BP＝データバッファ領域の先頭アドレス

出　　力　CF＝終了条件　0：正常終了　1：異常終了
AH＝ステータス情報（p.267:ステータス情報一覧表を参照）

機　　能　デバイスアドレス番号，ヘッド番号，シリンダ番号で指定された1トラックをフォーマットします．トラックの構造は図のようになっています．

インデックス	インデックスギャップ	セクタ 1	セクタ 2	…	セクタ n	インデックスギャップ

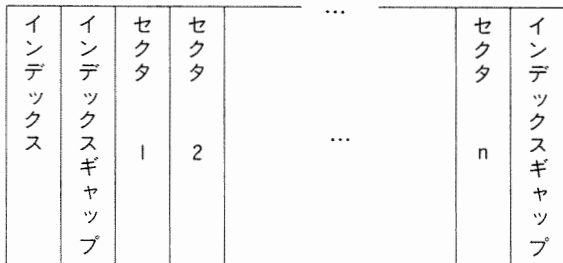



図 2-35　トラックの構造

フォーマットとは，トラック上にこのような構造を作ることです．PC-9801 シリーズのディスクのフォーマットは IBM フォーマットが採用されていますので，インデックス，インデックスギャップ，セクタ，トラックギャップの構造や形式は記録方式（FM（単密度）または MFM（倍密度））によって，決められています．

表 2-35　フォーマットの形式とギャップ長

ディスクタイプ	記録方式	セクタ長	セクタ当りのデータ数	トラック当りのセクタ数	ギャップ長
1MB	FM（単密度）	0	128 バイト	26	1BH
		1	256	15	2AH
		2	512	8	3AH
	MFM（倍密度）	1	256	26	36H
		2	512	15	54H
		3	1024	8	74H
640KB	FM	0	128	16	1BH
		1	256	9	2AH
		2	512	5	3AH
	MFM	1	256	16	33H
		2	512	9	50H
		3	1024	5	74H

このフォーマットの BIOS は，各セクタに書き込む ID 情報を任意に指定することができます．この ID は，シリンダ番号 C，ヘッド番号 H，セクタ番号 R，セクタ長 N，の4バイトで構成されています．よって，1トラック当たりに n セクタをフォーマットするときは，4 × n バイトのデータバッファ領域を確保して，その領域に ID データを格納する必要があります．例えば，MS-DOS の 1MB フォーマットの場合

には，n=8になるので，4×8=32バイトのデータバッファ領域を確保し，IDデータを格納します．IDデータの格納の形式は，表を参照してください．このBIOSをコールするときにはあらかじめバッファの領域を確保しておき，IDデータを格納してからにしてください．

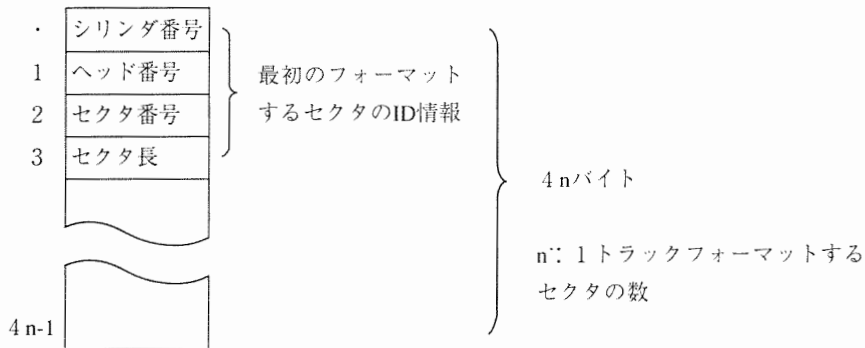



図2-36　IDデータの格納形式

主なシステムのフォーマットの形式を表にしました．参考にしてください．

ディスクタイプ	システム	記録方式	セクタ長
1MB	MS-DOS	MFM	3
	CP/M	MFM	1
	IMB-PC	MFM	2
640KB	MS-DOS	MFM	2
	CP/M	MFM	1

サンプル　このプログラムは2HDフロッピーディスクを，MS-DOSのフォーマット形式にしたがって物理フォーマットします．このディスクをMS-DOSからアクセスすると，初めは，「このフロッピーを使うことができません」等のメッセージが出てしまい認識してくれませんが，リトライをすると認識をしてくれます．これは，論理フォーマット（FAT（ファイル・アロケーション・テーブル）の書き込み）を施していないために起きてしまう現象です．

MS-DOS 1MB 物理フォーマットの形式

記録方式	MFM
セクタ長	3
シーケンス	1
シリンダ番号	0〜76
セクタ当たりのデータ数	1024
トラック当たりのセクタ数	8
ギャップ長	74H

```
#include <stdio.h>
#include <dos.h>

#define AH              0x7d    /* 01111101                                  */
#define DA_UA           0x90    /* 1MBインターフェース・ユニット番号0 */
#define SECTOR_N        3       /* セクタ長  3                               */
#define SECTOR_TRACK    8       /* トラック当たりのセクタ数  8               */
#define MAX_HEAD        1       /* 最大ヘッド数                              */
#define MAX_CYLINDER    76      /* 最大シリンダ数                            */
#define DATA_PATTERN    0       /* データ部の書き込みパターン                */

void format_track(int head,int cylinder)
{
    /* 引数によって指定されたトラックのフォーマットをします */

    struct REGPACK regs;

    char data[26 * 4];     /* IDデータ用のデータバッファの確保 */
    int i;

    regs.r_es = FP_SEG(data);
    regs.r_bp = FP_OFF(data);

    for (i = 0;i < SECTOR_TRACK;i ++){
        data[4 * i    ] = cylinder;
        data[4 * i + 1] = head;
        data[4 * i + 2] = i + 1;     /*  論理セクタ番号  */
        data[4 * i + 3] = SECTOR_N;
    }

    regs.r_ax = AH<<8 | DA_UA;
    regs.r_bx = SECTOR_TRACK * 4;
    regs.r_cx = SECTOR_N<<8 | cylinder;
    regs.r_dx = head<<8 | DATA_PATTERN;

    intr(0x1b,&regs);

    if (regs.r_flags & 1 == 1){
        printf("エラーのため実行を中止します¥n");
        exit(1);
    }
}

void main(void)
{
int head;
int cylinder;

    printf("このサンプルプログラムは¥n");
    printf("2HDディスクをMS-DOSフォーマット形式にしたがって¥n");
    printf("1MBフォーマットするものです¥n");
    printf("準備ができたら何かキーを押してください¥n");

    getch();

    for (cylinder = 0;cylinder <= MAX_CYLINDER;cylinder ++){
```

```
        for (head = 0;head <= MAX_HEAD;head ++){
            printf("ヘッド %d の %d セクタをフォーマット中です¥n¥x1bM",head,cylinder);
            format_track(head,cylinder);
        }
    }
    printf("¥nフォーマットを終了しました¥n");
}
```

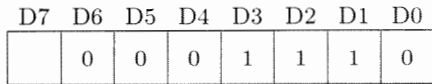

13 動作モードの設定 FDD

(640KB／1MB両用インターフェース時の1MBモード専用)

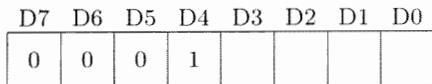
機能コード 0EH

割り込み INT 1BH

入力 AH=BIOSコマンド識別コード: 0EHまたは8EH

D7	D6	D5	D4	D3	D2	D1	D0
	0	0	0	1	1	1	0

D7: 0: 片面／両面モード指定
1: 48tpi (40シリンダ) ／96tpi (80シリンダ) モード指定

AL=各ユニットの動作情報

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	1				

D0: ユニット番号0
D1: ユニット番号1
D2: ユニット番号2
D3: ユニット番号3

AH	ALの各ユニット番号のBITの0／1設定	
0EH	0	片面モード
	1	両面モード
8EH	0	48tpi (40シリンダ) モード
	1	96tpi (80シリンダ) モード

出力 CF=終了条件　0：正常終了　1：異常終了
AH=ステータス情報 (p.267:ステータス情報一覧表を参照)

機能 640KB／1MB両用インターフェースを1MBモードで使用して，640KBのフロッピーディスクをアクセスする際の動作モードを設定します．

サンプル 640KBインターフェースのユニット番号0~3の動作モードを，48tpiモードに設定します．

```
#include <stdio.h>
#include <dos.h>

#define    AH        0x8e        /*  10001110                          */
#define    AL        0x10        /*  ユニット番号0〜3を48tpiモードに設定  */

void main(void)
{
    union REGS inregs,outregs;

    inregs.h.ah = AH;
    inregs.h.al = AL;

    int86(0x1b,&inregs,&outregs);

    printf("%x %x¥n",outregs.x.cflag,outregs.h.ah);

    if (outregs.x.flags & 1 == 1){
        printf("エラーです¥n");
    }
    else{
        printf("設定しました¥n");
    }

}
```

14 シーク FDD

機能コード **10H**

割り込み **INT 1BH**

入　　力 **AH=BIOS コマンド識別コード: 10H または 30H**

D7	D6	D5	D4	D3	D2	D1	D0
0	0	$\bar{r}$	1	0	0	0	0

$\bar{r}$ — 0:リトライあり（8回まで）　1:リトライなし

AL=デバイスアドレス番号 (p.264)
CL=開始シリンダ番号

出　　力 **CF=終了条件　0：正常終了　1：異常終了**
AH=ステータス情報（p.267:ステータス情報一覧表を参照）

機　　能 指定されたデバイスアドレス番号の装置のヘッドを，指定されたシリンダ番号まで移動させます．リード／ライト系のBIOSコマンドにもシーク機能がついており，シークを要求するフラグがON (1) のときにはシーク動作を伴い，OFF (0) のときには現在のシリンダを対象とした処理をします．

サンプル 1MBインターフェース，ユニット番号0の装置のヘッドを，76シリンダまで移動さ

せます.

```
#include <stdio.h>
#include <dos.h>

#define AH          0x10    /*  00010000                */
#define DA_UA       0x90    /*  1MB・ユニット番号0       */
#define CYLINDER    76      /*  シリンダ番号76に移動させる*/

void main(void)
{
    union REGS inregs,outregs;
    inregs.h.ah = AH;
    inregs.h.al = DA_UA;
    inregs.h.cl = CYLINDER;
    int86(0x1b,&inregs,&outregs);

    if (outregs.x.flags & 1 == 1){
        printf("エラーです¥n");
    }
    else{
        printf("正常に終了しました¥n");
    }
}
```

15 モーター停止モードの設定 FDD

（640KB／1MB 両用インターフェース専用）

機能コード 83H

割り込み INT 1BH

入　力 AH=BIOS コマンド識別コード: 83H

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	1	1

AL=デバイスアドレス番号 (p.264)

出　力 CF=終了条件　0：正常終了　1：異常終了
AH=ステータス情報（p.267:ステータス情報一覧表を参照）

機　能 フロッピーディスク装置のモーターを自動的に ON／OFF になるように設定をします．一度この設定をすると，リセットをするまで，常時 ON の状態（設定前の状態）に戻すことはできません．

サンプル 論理デバイス番号 90h のモーターを停止モードにするプログラムです．

```
#include <stdio.h>
#include <dos.h>
```

```
#define    AH       0x83        /*  10000011              */
#define    DA_UA    0x90        /*  1MB・ユニット番号0    */

void main(void)
{
    union REGS inregs,outregs;
    inregs.h.ah = AH;
    inregs.h.al = DA_UA;
        int86(0x1b,&inregs,&outregs);
    if (outregs.x.flags & 1 == 1){
        printf("エラーです¥n");
    }
    else{
        printf("モーター停止モードにしました¥n");
    }

}
```

■2-9-4 ハードディスク

●ハードディスクBIOSの一般形式

割り込み INT 1BH

ハードディスクBIOSのソフトウェア割り込みは，INT 1BHをつかいます．

入　力
AH＝BIOSコマンド識別コード
AL＝デバイスアドレスコード（DA：Device　Address）
　／ユニット番号（UA：Unit　Address）
ES：BP＝データバッファ領域の先頭アドレス
CX,DH,DL＝シリンダ，ヘッダ，セクタの指定

◆AH＝BIOSコマンド識別コード

レジスタAHの下位の4ビット（0〜3ビット）でコマンド識別コードを指定します．第5ビットでそのコマンドによって，SEEK動作の選択，リトライ動作の有無の選択が可能です．

コマンドに該当しないものを指定したときには，正常終了をします．

◆AL＝デバイスタイプ識別コード（DA：Device　Address）
　／ユニット番号（UA：Unit　Address）

UAは，ハードディスクの場合，ID番号を示します．DA／UAの関係は表を参照してください（p.264）．

◆絶対アドレスによるアクセスの場合のCX,DH,DL

CX＝シリンダ番号

DH＝ヘッド番号

DL＝セクタ番号

◆相対アドレスによるアクセスの場合のCX,DH,DL

相対アドレス＝｛（HDDのヘッド数）×CX＋DH｝×1トラック当たりのセクタ数＋DLの計算式で求めた値を，DL，CH，CLの3バイトに，つまりDLの最上位ビットからCLの最下位ビットの24ビットに格納する．

たとえば，シリンダ7，ヘッド2，セクタ8の相対アドレスを計算すると（HDDのヘッド数を4，1トラック当たりのセクタ数を33とします），

CX＝7

DH＝2

DL＝8

となるので，

相対アドレス＝　（4×7＋2）×33＋8＝0003E6H

したがって，

DL＝00H

CH＝03H

CL＝E6H

となります．

■ハードディスクBIOS一覧表（INT　OBH）

機能コード	機能	SASI,IDE	SCSI
01H	ベリファイ	○	○
03H	初期化	○	○
04H	センス	○	○
05H	データの書き込み	○	○
06H	データの読み出し	○	○
07H	シリンダ0へのシーク	○	○
08H	代替トラックの相対アドレス	○	×
0BH	代替トラックの割り付け	○	×
0DH	トラックのフォーマット	○	○
0FH	リトラクト	○	○

■ハードディスクBIOSステーター覧表

ＡＨ	ＣＦ	説明
００Ｈ	0	正常終了
０８Ｈ	0	エラー訂正されたデータを読んだ
２０Ｈ	1	バッファ領域のアドレスがＤＭＡバンクにまたがっている
３０Ｈ	1	一回でできるデータの転送容量を超えたデータ長を指定した
３８Ｈ	1	アクセスしたセクタが最大セクタアドレスを超えている
４０Ｈ	1	装置に異常があった
６０Ｈ	1	装置がＲｅａｄｙ状態でない
８８Ｈ	1	直接代替トラックをアクセスした
９０Ｈ	1	処理が一定時間内に終わらなかった
Ａ０Ｈ	1	ＩＤを読みだしたときに，ＣＲＣエラーが発生した
Ｂ０Ｈ	1	データを読みだしたときに，ＣＲＣエラーが発生した．
Ｃ０Ｈ	1	指定したセクタが，トラック上になかった
Ｃ８Ｈ	1	シークのエラー
Ｄ０Ｈ	1	不良ブロックを見つけた
Ｅ０Ｈ	1	ＩＤ検出後，ＤＡＭを検出できなかった
Ｆ０Ｈ	1	データを読み込むときにＤＡＭを検出できなかった

1　ベリファイ（VERIFY）　HDD

機能コード　01H

割り込み　INT　1BH

入　力　AH＝BIOSコマンド識別コード

D7	D6	D5	D4	D3	D2	D1	D0
0	0	$\overline{r}$	0	0	0	0	1

D5: 0：リトライあり（8回まで）
1：リトライなし

AL＝デバイスアドレス番号（DA／UA）

BX＝データブロック数（n）256×nバイト

●絶対アドレス指定

CX＝シリンダ番号

DH＝ヘッド番号

DL＝開始セクタ番号

●相対アドレス指定

DL，CH，CL24ビット＝相対アドレスの値

ES：BP＝データバッファ領域の先頭アドレス

出　力 **CF＝終了条件**

0：正常終了

1：異常終了

AH＝ステータス情報（p.292：ステータス情報一覧表を参照）

機　能 指定したデバイスアドレス番号，シリンダ番号，ヘッド番号に対応するトラックの開始セクタから，もしくは，相対アドレスで指定したセクタから指定したデータブロック数（256×nバイト）のデータを読み取り，データの読み取り動作ができることを確認するためのものです．データはメモリに格納しません．

サンプル ハードディスクデータのベリファイをするプログラムです．SCSIハードディスクの，シリンダ番号10，ヘッド番号3，開始セクタを5として2セクタ分だけデータをベリファイします．

```
#include <stdio.h>
#include <dos.h>

#define AH            0x01            /* 00000001          */
#define DA_UA         0xa0            /* SCSI絶対アドレス指定 */
#define CYLINDER      10              /* シリンダ番号10      */
#define HEAD          3               /* ヘッド番号    3     */
#define SECTOR_START  5               /* 開始セクタ    5     */
#define SECTOR_N      512             /* セクタ長      512   */
#define BYTE          SECTOR_N * 2    /* 512*2 バイト読みだす */

void main(void)
{
      struct REGPACK regs;

      unsigned char buf[BYTE];      /* 2セクタ分のバッファを確保 */

      int i;

      for (i = 0;i < BYTE;i ++)          /* バッファをクリア */
            buf[i] = 0;

      regs.r_ax = AH<<8 | DA_UA;
      regs.r_bx = BYTE;
      regs.r_cx = CYLINDER;
      regs.r_dx = HEAD<<8 | SECTOR_START;
      regs.r_es = FP_SEG(buf);       /* バッファのセグメントを得る */
      regs.r_bp = FP_OFF(buf);       /* バッファのオフセットを得る */
```

```
        intr(0x1b,&regs);

        if (regs.r_flags & 1 == 1){
              printf("異常終了です\n");
        }

        else{
              for(i = 0;i < 2;i ++)
                    printf("sector = %d [0] = %x [1023] = %x\n",
                          SECTOR_START + i , buf[i * SECTOR_N],
                          buf[(i + 1) * SECTOR_N - 1]);

              printf("正常に終了しました\n");
        }
}
```

2 初期化 (INITIALIZE) HDD

機能コード 03H

割り込み INT 1BH

入　力 AH＝03H

AL＝デバイスアドレス番号 (DA／UA)

出　力 CF＝終了条件

0 ：正常終了

1 ：異常終了

AH＝ステータス情報 (p292：ステータス情報一覧表を参照)

機　能 指定したデバイスアドレス番号，を初期化します．この時，以下の処理を行います．

- FDCを初期化する
- ハードディスクの接続をチェックする
- シリンダ0にシークさせる
- リトラクト処理を行う

SCSIハードディスクのイニシャライズをします．電源投入時と同じ様なことをしますので，実行するとしばらくしてから反応が返ってきます．

サンプル

```
#include <stdio.h>
#include <dos.h>

#define AH      0x03      /*  00000011        */
#define DA_UA   0xa0      /*  SCSI絶対アドレス指定  */

void main(void)
{
```

```
	union REGS inregs,outregs;
	inregs.h.ah = AH;
	inregs.h.al = DA_UA;
		int86(0x1b,&inregs,&outregs);
	if (outregs.x.flags & 1 == 1){
		printf("エラーです\n");
	}
	else{
		printf("初期化しました\n");
	}

}
```

3 センス（SENSE） HDD

機能コード 04H

割り込み INT 1BH

入　力 AH＝04H または 84H

AL＝デバイスアドレス番号（DA／UA）

出　力 CF＝終了条件

0 ：正常終了

1 ：異常終了

AH＝ステータス情報（p.292：ステータス情報一覧表を参照）

以下は，呼び出した時に，AH＝84Hを指定したときのみ有効

BX＝セクタ長

CX＝シリンダ数

DH＝ヘッド数

DL＝セクタ数

機　能 指定したデバイスアドレス番号の状態を得ることができます．

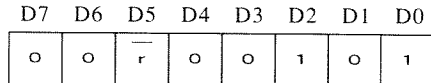
サンプル SCSIハードディスクの状態を調べます．

```
#include <stdio.h>
#include <dos.h>

#define  AH        0x84      /*  10000100          */
#define  DA_UA     0xa0      /*  SCSI絶対アドレス指定  */

void main(void)
{
	union REGS inregs,outregs;
```

```
        inregs.h.ah  =  AH;
        inregs.h.al  =  DA_UA;
        int86(0x1b,&inregs,&outregs);
        printf("セクタ長    %d\n",outregs.x.bx);
        printf("シリンダ数  %d\n",outregs.x.cx);
        printf("ヘッド数    %d\n",outregs.h.dh);
        printf("セクタ数    %d\n",outregs.h.dl);
        printf("ax =        %x\n",outregs.x.ax);
}
```

4 データの書き込み（WRITE DATA） HDD

機能コード 05H

割り込み INT 1BH

入　力 AH＝BIOSコマンド識別コード

AL＝デバイスアドレス番号（DA／UA）

BX＝データブロック数（n）256×nバイト

●絶対アドレス指定

CX＝シリンダ番号

DH＝ヘッド番号

DL＝開始セクタ番号

●相対アドレス指定

DL，CH，CL＝相対アドレスの値（24ビット）

ES：BP＝データバッファ領域の先頭アドレス

出　力 CF＝終了条件

0：正常終了

1：異常終了

AH＝ステータス情報（p.292：ステータス情報一覧表を参照）

機　能 データバッファ領域のデータを，指定されたデータブロック数の長さ（256×nバイト）分だけ，指定したデバイスアドレス番号，シリンダ番号，ヘッド番号に対応するト

ラックの開始セクタから，もしくは相対アドレスで指定したセクタから書き込みます．

5 データの読みだし（READ　DATA）

機能コード 06H

割り込み INT　1BH

入　　力 AH＝BIOSコマンド識別コード

AL＝デバイスアドレス番号（DA／UA）

BX＝データブロック数(n) 256×nバイト

●絶対アドレス指定

CX＝シリンダ番号

DH＝ヘッド番号

DL＝開始セクタ番号

●相対アドレス指定

DL，CH，CL24ビット＝相対アドレスの値（24ビット）

ES：BP＝データバッファ領域の先頭アドレス

出　　力 CF＝終了条件

0：正常終了

1：異常終了

AH＝ステータス情報（p.292：ステータス情報一覧表を参照）

機　　能 指定したデバイスアドレス番号，シリンダ番号，ヘッド番号に対応するトラックの開始セクタから，もしくは，相対アドレスで指定したセクタから指定したデータブロック数（256×nバイト）のデータをデータバッファ領域の先頭アドレスに読み込みます．

サンプル ハードディスクデータの読みだしをするプログラムです．SCSIハードディスクから，シリンダ番号10，ヘッド番号3，開始セクタを5として2セクタ分だけデータをよみだします．

```
#include <stdio.h>
#include <dos.h>

#define AH             0x06
#define DA_UA          0xa0              /* scsi絶対アドレス          */
#define CYLINDER       10                /* シリンダ番号10            */
#define HEAD           3                 /* ヘッド番号     3          */
#define SECTOR_START   5                 /* 開始セクタ     5          */
#define SECTOR_N       512               /* セクタ長       512        */
#define BYTE           SECTOR_N * 2      /* 512*2 バイト読みだす      */

void main(void)
{
      struct REGPACK regs;

      unsigned char buf[BYTE];        /* 2セクタ分のバッファを確保 */
      int i;

      for (i = 0;i < BYTE;i ++)        /* バッファをクリア */
            buf[i] = 0;

      regs.r_ax = AH<<8 | DA_UA;
      regs.r_bx = BYTE;
      regs.r_cx = CYLINDER;
      regs.r_dx = HEAD<<8 | SECTOR_START;
      regs.r_es = FP_SEG(buf);        /* バッファのセグメントを得る */
      regs.r_bp = FP_OFF(buf);        /* バッファのオフセットを得る */

      intr(0x1b,&regs);

      if (regs.r_flags & 1 == 1){
            printf("異常終了です\n");
      }

      else{
            for(i = 0;i < 2;i ++)
                  printf("sector = %d [0] = %x [1023] = %x\n",
                        SECTOR_START + i , buf[i * SECTOR_N],
                        buf[(i + 1) * SECTOR_N]);

            printf("正常に終了しました\n");
      }
}
```

6 シリンダ0へのシーク (RECALIBRATE) HDD

機能コード 07H

割り込み INT 1BH

入 力 AH＝BIOSコマンド識別コード

D7	D6	D5	D4	D3	D2	D1	D0
0	0	r̄	0	0	1	1	1

r̄：0：リトライあり（8回まで）
1：リトライなし

AL＝デバイスアドレス番号（DA／UA）

出 力 CF＝終了条件

0：正常終了

1：異常終了

AH＝ステータス情報（ステータス情報一覧表を参照）

機 能 指定したデバイスアドレス番号の装置のヘッドをシリンダ0へシークさせます．

サンプル SCSIハードディスクで，0シリンダへのシークコマンドを実行します．

```
#include <stdio.h>
#include <dos.h>

#define  AH          0x07         /*  00000111               */
#define  DA_UA       0Xa0         /*  SCSI絶対アドレス指定      */

void main(void)
{
      union REGS inregs,outregs;
      inregs.h.ah = AH;
      inregs.h.al = DA_UA;
      int86(0x1b,&inregs,&outregs);
      if (outregs.x.flags & 1 == 1){
            printf("エラーです\n");
      }
      else{
            printf("正常に終了しました\n");
      }

}
```

Assign Alternate Track　コマンド

機能コード 08H

割り込み INT　1BH

入　力 AH＝0BH

AL＝デバイスアドレス番号（DA／UA）

BX＝データブロック数

●絶対アドレス指定

CX＝シリンダ番号

DH＝ヘッド番号

DL＝セクタ番号

ES：BP＝データバッファ領域の先頭アドレス（4バイトが必要）

出　力 CF＝終了条件

0：正常終了

1：異常終了

AH＝ステータス情報（p.292：ステータス情報一覧表を参照）

機　能 指定した絶対アドレスから相対アドレスの値を計算して，その値をデータバッファ領域の先頭アドレスに格納します．

サンプル SASIハードディスクで，シリンダ番号7，ヘッド番号2，セクタ番号8，の相対アドレスを計算します．なお，ここで使用するハードディスクのヘッド数は4，1トラック当りのセクタ数は33，セクタ長は256とします．

```
#include  <stdio.h>
#include  <dos.h>

#define  AH            0x08          /*  00001011                  */
#define  DA_UA         0x80          /*  sasi絶対アドレス            */
#define  CYLINDER      7             /*  シリンダ番号   7            */
#define  HEAD          2             /*  ヘッド番号     2            */
#define  SECTOR_START  8             /*  開始セクタ     8            */
#define  SECTOR_N      256           /*  セクタ長       256          */
#define  BYTE          0

void  main(void)
{
        struct  REGPACK  regs;
```

```
    unsigned char buf[3];                    /* バッファを確保 */
    int i;

    for (i = 0;i < 4;i ++)                   /* バッファをクリア */
        buf[i] = 0;

    regs.r_ax = AH<<8 | DA_UA;
    regs.r_bx = BYTE;
    regs.r_cx = CYLINDER;
    regs.r_dx = HEAD<<8 | SECTOR_START;
    regs.r_es = FP_SEG(buf);                 /* バッファのセグメントを得る */
    regs.r_bp = FP_OFF(buf);                 /* バッファのオフセットを得る */

    intr(0x1b,&regs);

    if (regs.r_flags & 1 == 1){
        printf("異常終了です\n");
    }

    else{
        for(i = 0;i < 4;i ++)
            printf("buf[%d]= %x\n",i,buf[i]);
        printf("正常に終了しました\n");
    }
}
```

8 Format Alternate Track コマンド HDD

機能コード 0BH

割り込み INT 1BH

入　　力 AH＝BIOSコマンド識別コード

0：リトライあり（8回まで）
1：リトライなし

AL＝デバイスアドレス番号（DA／UA）

BH＝インタリーブファクタ

●絶対アドレス指定

CX＝不良トラックの存在するシリンダ番号

DH＝不良トラックの存在するヘッド番号

DL＝セクタ番号

ES：BP＝データバッファ領域の先頭アドレス
（代替トラックのアドレスを指定）

出　力　**CF＝**終了条件
0：正常終了
1：異常終了

AH＝ステータス情報（p.292：ステータス情報一覧表を参照）

機　能　不良トラックに対して，代替トラックを指定します．HDDは，容量が大きいので不良セクタが1つや2つ存在してしまったためにHDD全体が不良となってしまうのは，もったいないことです．そこで，不良セクタのあるトラックは使わずに，その代わりとして代替トラックを指定することによって解消します．このコマンドでは，トラック内に不良セクタがあるとき，ID部内に不良フラグをセットし，データ部に代替トラックアドレスを書き込みます．以降，このトラックのアクセスは代替トラックへのアクセスに変換されます．このBIOSをコールするときは，代替トラックのアドレスは，データバッファ領域にあらかじめ入れておきます．代替トラックのアドレスのデータは，Assign Alternate Track コマンドを使って得ます．

9　トラックのフォーマット（FORMAT TRACK／DRIVE）　HDD

機能コード　**0DH**

割り込み　**INT　1BH**

入　力　**AH＝**BIOSコマンド識別コード

AL＝デバイスアドレス番号（DA／UA）

BH＝インタリーブファクタ

●絶対アドレス指定

CX＝トラック単位にフォマットする場合のシリンダ番号

DH＝トラック単位にフォーマットする場合のヘッド番号

DL＝0

●相対アドレス指定

DL，CH，CL＝相対アドレスの値（24ビット）

注）1：インタリーブファクタは通常5を指定する

2：デバイス単位にフォーマットするときは，CX＝0，DH＝0をする

出　力 **CF＝**終了条件

0：正常終了

1：異常終了

AH＝ステータス情報（p.292：ステータス情報一覧表を参照）

機　能 デバイスアドレス番号で指定される装置に対して，トラック単位，またはデバイス単位でフォーマットをします．

10 リトラクト（RETRACT） HDD

機能コード **0FH**

割り込み **INT　1BH**

入　力 **AH＝**BIOSコマンド識別コード

AL＝デバイスアドレス番号（DA／UA）

出　力 **CF＝**終了条件

0：正常終了

1：異常終了

AH＝ステータス情報（p.292：ステータス情報一覧表を参照）

機　能 指定したデバイスアドレス番号のデバイスのヘッドを，シッピングゾーンに移動させます．ハードディスクのヘッドはフロッピーディスクとは異なり，ディスクの媒体が高速で回転しているため，ヘッドが直にディスク媒体に触れてしまうと傷がついてしまうので，そのヘッドをディスク媒体から少しだけ浮上させています．高速回転をしているときは，ディスク媒体は少しだけ浮上しているので問題はないのですが，ディスク媒体の回転数が，ある回転数よりも下がってくるとヘッドが直にディスク媒体に触れてしまい，傷を付けてしまいます．これでは，電源をきるたびにディスクの媒体を傷つけてしまい，ハードディスクは，高価な消耗品となってしまいます．これを防ぐために，ディ

スクの円周に，ヘッドを接触させてもよい領域（シッピングゾーン）を設けて，電源を切る前には，この領域にヘッドを移動させるようにします．最近のハードディスクは，ほとんどのものが電源が切れると，このシッピングゾーンにヘッドを自動的に移動させるような機能がついているため，この機能は必要ありません．

サンプル　SCSIハードディスクで，リトラクトコマンドを実行します．

```
#include <stdio.h>
#include <dos.h>

#define AH          0x0f        /*  00001111          */
#define DA_UA       0xa0        /*  SCSI絶対アドレス指定  */

void main(void)
{
        union REGS inregs,outregs;
        inregs.h.ah = AH;
        inregs.h.al = DA_UA;
        int86(0x1b,&inregs,&outregs);

        if (outregs.x.flags & 1 == 1){
                printf("エラーです\n");
        }
        else{
                printf("正常に終了しました\n");
        }
}
```

参考文献

・テクニカルデータブックBIOS編（ASCII出版）
・スーパーテクニック（同上）
・MSDOS基本プログラミング第2集 PC9801の割り込みとBIOS活用法（CQ出版社）
・ディスクBIOSとC言語（工学図書株式会社）

第二部 98各機能の標準的利用方法　§2・9 ディスク

§2-10 RS-232C

98には，シリアルI/O用LSIとして，μPD8251A（または相当品）が内蔵されています．これにより，RS-232Cによる他の機器や，他のコンピュータとの通信ができます．

通信方式には，同期式と調歩同期式（非同期式）があり，同期式は送受信双方が同期して，データ通信を行います．それに対して，調歩同期式はデータの始まりにスタートビットと呼ばれるビットを付加し，さらにデータの終わりにストップビットというビットを付加することによって，データの始まりと終わりを判断して通信を行います．一般にRS-232Cにおいて通信をするときには，調歩同期式が使われ，同期式が使われることはめったにありません．以下，調歩同期式を使用することを前提に説明をしていきます．

98におけるRS-232Cは，ハードウェアやBIOSの出来があまりよくありません．メーカー保証およびBIOSが対応しているのは，9600bpsとなっています．初代98が発売された当初は，これで十分だったのかもしれませんが，その仕様を今もなお引き継いでいるために，最近の9600bps以上の通信速度を持つモデムの出現など，高速通信に対応できなくなっています．しかしながら，システムクロックが10MHz系の機種では，メーカー保証範囲をはずれるものの，19200や38400bpsが利用できる機種が多くあります．また，最近の98では19200bpsまでメーカーが保証しているものもあります．

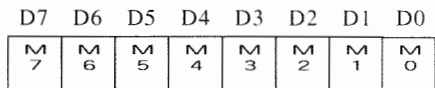
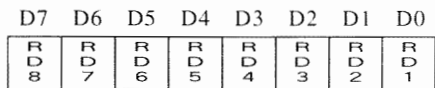
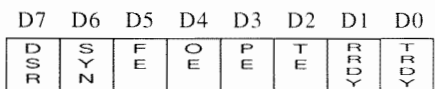
■2-10-1 RS-232CのI/Oポート

BIOSでは，送信割り込みのサポート，ハードウェアフロー制御がないことや，システムクロック10MHz系の機種で最高通信速度が9600bpsという制限が付くことから，RS-232C関係のLSIを直接制御して（BIOSを使わずに）通信する方法が，多く利用されています．ここでは，そのようなプログラムを書くために必要な事項を含め，RS-232CのI/Oについて説明します．

RS-232Cを制御するには，シリアルI/O用のLSIを制御する以外に，通信速度決定のためのタイマの設定や，送受信割り込みの設定やRS-232Cの信号線の読み出しなどに関係する，システムポートの操作，さらに，割り込みコントローラの設定が必要になります．これら必要なI/Oポートの一覧を表2-36に示します．なお，割り込みコントローラおよびタイマの詳しい操作法法については，ここでは省略します．詳しくは，「1-5．割り込み」，「2-3.タイマ」を参照してください．

表2-36　RS-232C関係I/Oポート

リード／ライト	I／O アドレス	命令（機能）	D7	D6	D5	D4	D3	D2	D1	D0
リード	02H	IMRリード（8259割り込みマスク読込）	M7	M6	M5	M4	M3	M2	M1	M0
	30H	リードデータ（232cからのデータ読込）	RD8	RD7	RD6	RD5	RD4	RD3	RD2	RD1
	32H	ステータス（232cの信号線の状態読込）	DSR	SYN	FE	OE	PE	TE	RRDY	TRDY
	33H	リードシグナル（232cの信号線の状態読込）	—CI	—CS	—CD	*	*	*	*	*
ライト	00H	OCW2（8259 EOI）	R	SL	EOI	0	0	L2	L1	L0
	02H	OCW1（8259割り込みマスク設定）	M7	M6	M5	M4	M3	M2	M1	M0
	30H	データライト（232cへデータ送信）	SD8	SD7	SD6	SD5	SD4	SD3	SD2	SD1
	32H	モード(A)(調歩同期)（8251動作モード設定）	S2	S1	EP	PEN	L2	L1	B2	B1
	32H	モード(B)(同期)（8251動作モード設定）	SCS	ESD	EP	PEN	L2	L1	0	0
	32H	コマンド（8251の設定）	EH	IR	RTS	ER	SBR	RXE	DTR	TXE
	35H	マスクセット（8251の割り込みマスク設定）	*	*	*	*	*	TXRE	TXEE	RXRE
	75H	カウンタセット（タイマーの値設定）（転送速度設定）	C7 ------------------------------------ C0 C15 ---------------------------------- C8							
	77H	カウンタモード（タイマのモード指定）	SC1	SC0	RL1	RL0	M2	M1	M0	BCD

●IMRリード/OCW1

●リードデータ

●ステータス

◆パリティエラー

付加されたエラー検出のためのビットにより，データのエラーが検出された場合にこのエラーとなります．

◆オーバーランエラー

次のデータが到着し受信バッファに入るときに，1つ前のデータを受信バッファからCPUが読みだしていない場合にこのエラーとなります．

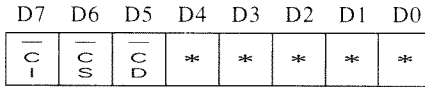
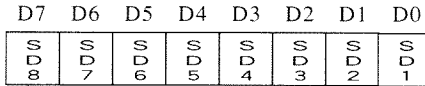
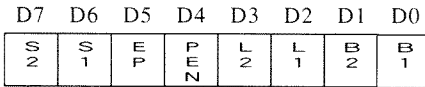
◆フレーミングエラー

ストップビットが検出されないときにこのエラーとなります．

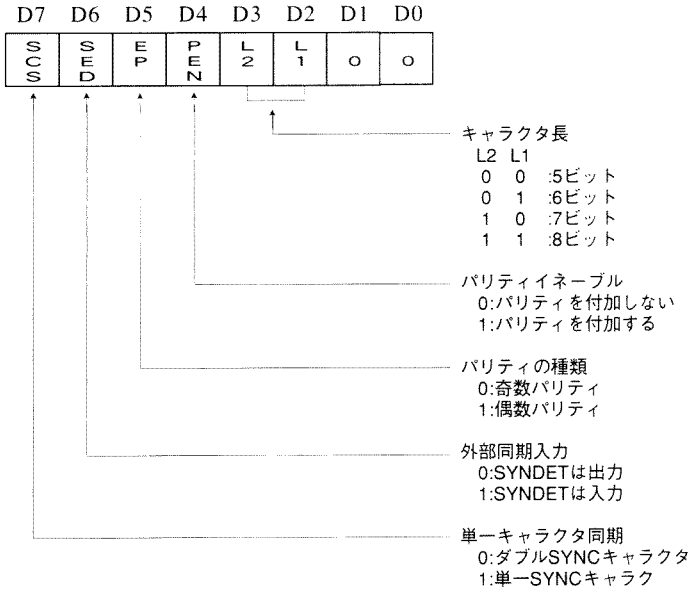
◆ブレーク信号

これは，受信または送信端子が長時間0に保たれる特殊な信号です．通常は，調歩同期式の場合，1データ単位で制御用のビットが入るので，長時間0になることはありません．ブレーク信号は，特殊な制御に利用されます．

※ブレーク信号以外のエラーフラグは，エラークリアしない限り1に保たれます．なお，エラーフラグが1になっても，その後の通信には関係ありません．

●リードシグナル

●データライト

●モード（A）（調歩同期）

◆通信速度の決定方法（調歩同期の場合）

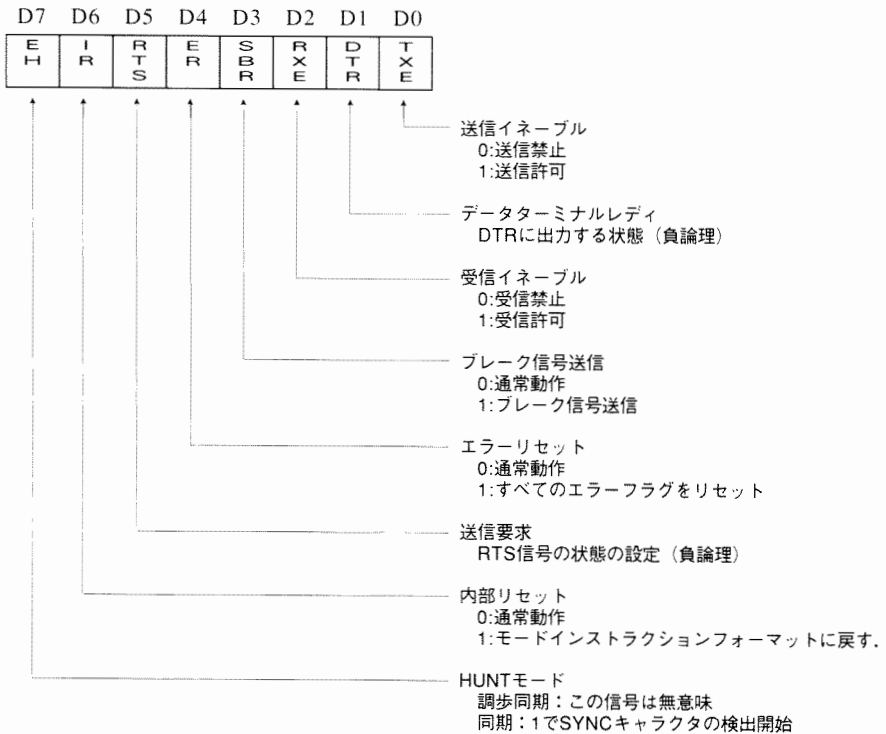
調歩同期式の場合，B2，B1で，×16または×64モードを使用します．このモードとタイマLSIに設定した値で通信速度が決定されます．タイマの設定は，232CのLSIにどれくらいの周波数のクロックを供給するかという設定です．モードの設定は供給されるクロックを何分周して通信速度の設定に利用するかという設定です．ここでのモード設定は，どちらでも構わないのですが，それぞれのモードによって，目的の通信速度を得るためにタイマLSIに設定する値が違ってきます．タイマLSIに設定する値の詳細および，タイマLSIに値を設定する方法は，「2-3.タイマ」を参照してください．

●モード（B）（同期）

◆通信速度の決定方法（同期の場合）

同期式の場合，タイマLSIに設定した値で通信速度が決定されます．

タイマの設定は，232CのLSIにどれくらいの周波数のクロックを供給するかという設定です．

目的の通信速度によって，タイマLSIに設定する値が違ってきます．タイマLSIに設定する値の詳細および，タイマLSIに値を設定する方法は，「2-3.タイマ」を参照してください．

●コマンド

◆モードとコマンド設定

モード設定用のモードレジスタとコマンド用のコマンドレジスタのI/Oポートアドレスは同じ32Hになっています．最初にモードレジスタを設定し，次にコマンドレジスタを設定するという順です．一度モードレジスタを設定すると，以後，I/Oアドレス32Hはコマンドレジスタとなります．ただし，コマンドレジスタのbit6（内部リセット）に1を指定するとリセットし，モード設定に戻ります．

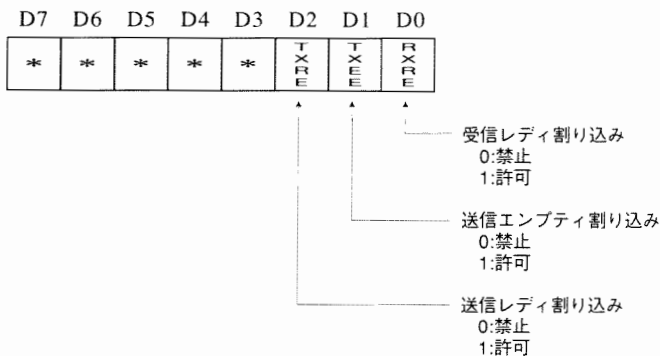
上記の理由からも，I/Oポート32Hに，00Hを3回，40Hを1回出力すれば，I/Oポート32Hがモードレジスタであったとしても，コマンドレジスタであったとしても232CのLSI（8251）を確実にリセットすることができます．232CのLSIを直接操作するときは，まず最初にこの方法でリセットするとよいでしょう．

◆サンプルプログラム

10MHz系の機種で，BIOSでは実現できない38400bps，8bit，no parity，stopbit 1 というパラメータで初期化します．

```
mov     dx,32h          ;232cのLSI(8251)をリセット
mov     al,0
out     dx,al
jmp     $+2
out     dx,al
```

```
jmp     $+2
out     dx,al
jmp     $+2
mov     al,40h
out     dx,al

mov     al,10110110b     ;通信速度設定（タイマーセット）
out     77h,al
mov     al,4             ;タイマに4を設定
jmp     $+2
out     75h,al
mov     al,0
jmp     $+2
out     75h,al

mov     al,4eh           ;8251モードセット(モードは1/16)
out     32h,al
jmp     $+2
mov     al,00100111b     ;8251コマンドセット
out     32h,al
```

●マスクセット

◆受信レディ割り込み

232Cで受信したデータがCPUより読み取りが可能になったときに発生する割り込みです.

◆送信エンプティ割り込み

2つある送信バッファ(※)の両方が空になったときに発生する割り込みです.

◆送信レディ割り込み

第2送信バッファ(※)が空になったときに発生する割り込みです.

※送信バッファについて

232CのLSIには，送信バッファが2つあります．図2-37に示すように　直列のバッファとなっています.

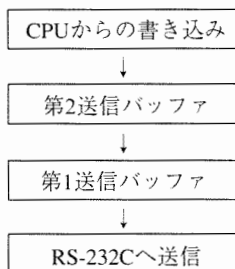



図2-37 2つの送信バッファの関係

CPUから書き込まれたデータは，最初に第2送信バッファに入ります．その後，第1送信バッファが空であれば，第1送信バッファに転送され，その後，送信可能であれば，送信されます．

つまり，データライトで送信データを書き込んだ場合，第2送信バッファに入り，送信されるデータは，第1送信バッファから読みだされます．第1から第2送信バッファへの転送はLSIが第1送信バッファが空になった時点で自動的に行います．

このように，232CのLSIはバッファを2バイト分持っていて，第2送信バッファが空になった時点（第1送信バッファが空でなくても）でCPUからデータが書き込めるために，効率のよい送信を行うことができます．

プログラムを作る上で送信割り込みを利用する場合，送信エンプティ割り込みではなく，送信レディ割り込みを利用したほうが効率がよくなるわけです．また，送信可能かどうかをステータスより判断するときにおいても，TEよりもTxRDYを参照したほうが効率がよいといえます．

※送受信両方の割り込みを利用する方法

98ではRS-232Cの送受信割り込みでは，同一の割り込み番号が割り当てられています．この送受信割り込みを両方利用するには，割り込みルーチン内に工夫が必要です．送信割り込み処理時のデータ受信の発生などの問題があるからです．これを簡単に解決する方法に，割り込みマスクを利用する方法があります．以下，簡単に割り込みルーチン内での具体的な処理方法を示します．

1. 送受信両方の割り込みマスクをする

```
mov al,0
out 35h,al      ;マスクセットに0を設定
```

2. 送受信どちらからの割り込みか判断し，受信なら受信処理，送信なら送信処理を行う．両方なら両方行う．

3. 割り込み処理を終了する時に，割り込みマスクを解除する．ただし，送信データがバッファに残っていなければ受信割り込みのみ許可する．

```
mov al,5     ;受信割り込みのみ許可の場合は，mov al,1
out 35h,al   ;割り込みマスク解除
```

これにより，たとえ送信処理中にデータを受信したとしても，送信割り込みが終了した時点で受信割り込みを発生させることが出来ます．割り込みマスクをはずすことにより，割り込みの立ち上がりエッジを作ることができるわけです．これは98の232cで利用されているLSIが，CPUからデータを読みださない限りRxRDY端子がLにならないという特性，および，この端子はシステムポートのLSIによりマスク出来るという特性を利用したものです．

■2-10-2 RS-232CのBIOS

RS-232CのI/Oポートで示したように，RS-232Cを利用して通信を行うまでには，多くの手続きが必要になります．そこで，簡単にRS-232Cが利用できるBIOSが用意されています．I/Oの直接制御に比べると，かなり少ない手続きで利用することができます．

しかしながら，RS-232CのBIOSは送信割り込みがサポートされてないことや，ハードウェアフロー制御がなく，これを実現するためには，フロー制御のためのプログラムを付加しなければならないという欠点があります．また，BIOSでサポートしているのは調歩同期式のみです．

RS-232Cを本格的に利用するには，少々物足りないといえるでしょう．しかし，かなり簡単にRS-232Cインターフェースを利用できるようになっているので便利であるといえます．

なお，基本的に内蔵ポートを対象にBIOSの説明をします．

■RS-232C BIOS一覧（INT　19H）

機能コード	機能	ノーマル	ハイレゾ
0 0 H	RS-232C BIOS 初期化　（シングルモード）	○	○
0 1 H	RS-232C BIOS 初期化（Xフロー制御あり）（シングルモード）	○	○
0 2 H	受信データ長の取得	○	○
0 3 H	データ送信	○	○
0 4 H	データ受信	○	○
0 5 H	8251へのコマンド出力	○	○
0 6 H	ステータスの取得	○	○
0 7 H	RS-232C BIOS 初期化（拡張モード）	×	○

RS-232C　BIOS初期化

割り込み INT　19H

入　力 AH←00H

AL←通信速度に対応した値

BH←送信時タイムアウト時間（500msec単位）

（00HでデフォルトトÓ02H（1sec）となる）

BL←受信時タイムアウト時間（500msec単位）

（00Hでデフォルトの1EH（15sec）となる）

（ハイレゾモードではデフォルトは3CH（30sec））

CH←8251モード設定データ

CL←8251コマンド設定データ

DX←受信バッファのサイズ（バイト単位）

ES：DI←受信バッファの先頭番地

出　力 AH→00H：正常終了

解　説 RS-232CのBIOSを初期化します．ALに設定する値は，表2-37に示します．

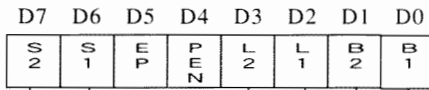
表2-37　ALに設定する値と通信速度の関係

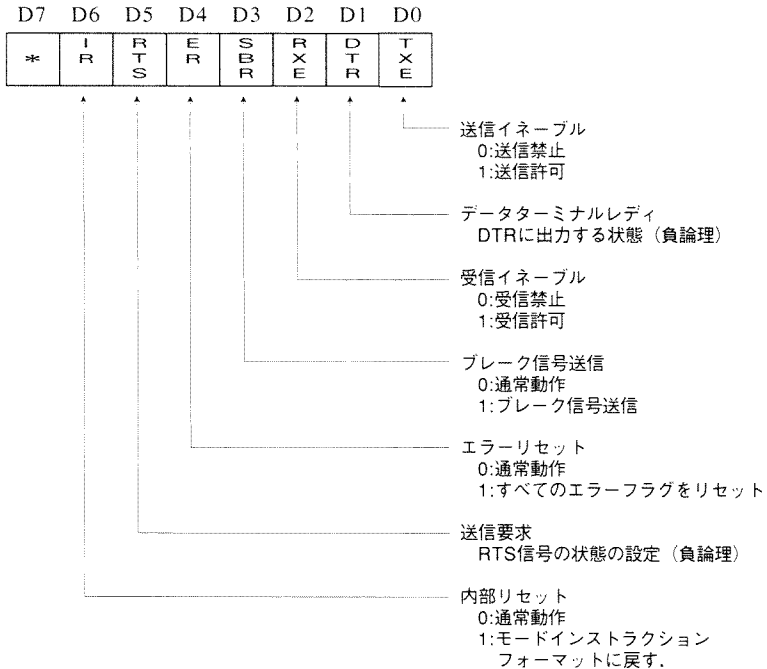
ＡＬに設定する値	00H	01H	02H	03H	04H	05H	06H	07H
通信速度(bps)	75	150	300	600	1200	2400	4800	9600

※ 08H以上の値を設定した場合は，1200bpsになります．

送信時タイムアウト時間とは，送信するときにTXRDYになっていない場合，TXRDYになるまで，最大どれくらい待つかという設定です．同様に，受信時タイムアウト時間とは，受信バッファが空であるときに読み出しを行った場合，RXRDY割り込みがかかるまで，どれくらい待つかという設定です．

CHに設定する8251モード設定データとは，8251のモードレジスタに設定する値そのものです．詳しくは，次頁のようになっています．

CH 8251モード設定データ

通信速度が9600bpsの場合，CHレジスタで設定する通信速度（B2,B1）はたとえ×64モードを指定したとしても，自動的に×16モードになります．なお，×16,×64モードのどちらを使用するかということですが，このCHレジスタに設定された値と，ALレジスタで指定した通信速度の値をBIOSが判断し，結果的にALで指定した通信速度になるように，BIOSがタイマなどに設定する値を自動的に決定します．つまり，×16，×64モードについては特に気にせず，どちらを指定しても構わないということです．

CLに設定する8251コマンド設定データとは，8251のコマンドレジスタに設定する値そのものです．詳しくは，次頁のようになっています．

ES:DIには，受信バッファの番地，DXには受信バッファのサイズを設定します．これで設定したバッファに受信したデータが，読みだされるまでの間貯えられます．なおDXに指定したバッファのサイズに対して，実際にはDX×2+4の大きさの領域を確保する必要があることに注意してください．バッファには，受信したデータのほかに，そのデータを受信したときのステータスが入るためにDXに指定した大きさより多くの領域が必要となります．

サンプル

4800bps,8 bit,no parity,stopbit 1 でRS-232C(BIOS)を初期化します．

```
#include <stdio.h>
#include <dos.h>

union REGS inregs, outregs;
struct SREGS segregs;
char buf[504];                          /* 受信バッファ */
                                            /* DX×2+4バイト確保する */

void main(void)
{
      inregs.h.ah = 0;        /* 232c初期化 */
      inregs.h.al = 6;        /* 4800bps */
      inregs.h.bh = 0;        /* 送信タイムアウトは筑湊拂 */
      inregs.h.bl = 0;        /* 受信タイムアウトは筑湊拂 */
      inregs.h.ch = 0x4f;     /* モード設定 */
      inregs.h.cl = 0x27;     /* コマンド設定 */
      inregs.x.dx = 250;      /* 受信バッファの長さ */
```

```
	inregs.x.di  = FP_OFF(buf);	/* バッファのアドレス */
	segregs.es  = FP_SEG(buf);	/* バッファのセグメント */
	int86x(0x19, &inregs, &outregs, &segregs);
	if(outregs.h.ah) {
		printf("初期化に失敗しました\n");
	}
}
```

2 RS-232C BIOS初期化(Xフロー制御あり)

割り込み INT 19H

入力 AH←01H

AL←通信速度に対応した値

BH←送信時タイムアウト時間（500msec単位）

（00Hでデフォルトの02H（1sec）となる）

BL←受信時タイムアウト時間（500msec単位）

（00Hでデフォルトの1EH（15sec）となる）

（ハイレゾモードではデフォルトは3CH（30sec））

CH←8251モード設定データ

CL←8251コマンド設定データ

ES：DI←受信バッファの先頭番地

DX←受信バッファのサイズ（バイト単位）

出力 AH→00H：正常終了

解説 RS-232CのBIOSを初期化します．AH=00Hの初期化とは，Xフロー制御がサポートされる点で違いますが，その外のパラーメータの設定方法は同じです．各パラメータの設定方法は，そちらを参照してください．

Xフロー制御は，次のような動作をします．

- 受信バッファに格納されたデータが，受信バッファの大きさの3/4になったら，13Hを送信し，いったん送信の停止要求を行います．
- 受信バッファに格納されたデータが，受信バッファの大きさの1/4になったら，11Hを送信し，送信再開の要求を行います．

3 受信データ長の取得

割り込み INT 19H

入　力 AH←02H

出　力 AH→00H：正常終了
01H：RS-232Cが初期化されていない
02H：受信バッファがオーバーフローしている
CX→受信データ長

解　説 受信バッファに，どれだけのデータが格納されているか調べます．

サンプル 受信データ長を表示します．ただし，このプログラムを実行するためには，RS-232C(BIOS)が初期化されていなければなりません．

```
#include  <stdio.h>
#include  <dos.h>

union  REGS  inregs,  outregs;

void  main(void)
{
       inregs.h.ah  =  2;
       int86(0x19,  &inregs,  &outregs);
       printf("受信データ長=%d\n",  outregs.x.cx);
}
```

4 データ送信

割り込み INT 19H

入　力 AH←03H
AL←送信データ
CL←参照ステータスビット（拡張モード（ハイレゾ）のみ）

出　力 AH→00H：正常終了
01H：RS-232Cが初期化されていない
02H：受信バッファがオーバーフローしている
03H：8251が送信可，受信可の状態でない．
05H：送信Xフロー制御がイネーブルのとき，XOFFを受信して送信をいったん停止している状態（ハイレゾのみ）

解　説　RS-232Cにデータを1バイト送信します．

送信するときに，シングルモードの場合はTxRDYのステータスのみを参照します．それ以外は，ユーザーが参照，操作する必要があります．

拡張モードでは，TxRDY以外にTxE，CS，DSRも参照することができます．また，どれを参照するかということを設定することができます．参照するステータスのビットを1にして，CLレジスタに設定します．CLレジスタに設定する値の詳細は，以下のようになっています．

D7	D6	D5	D4	D3	D2	D1	D0
DSR	CS	0	0	0	TxE	0	0

参照するステータスを1に設定します．

送信不可能の場合は，初期化コマンドで指定した送信時タイムアウト時間だけ待ちます．その間に送信可能にならなければ，エラーとなります．

サンプル　データを1バイト送信します．ただし，このプログラムを実行するためには，RS-232C(BIOS)が初期化されていなければなりません．

```
#include <stdio.h>
#include <dos.h>

union REGS inregs, outregs;

void main(void)
{
	inregs.h.ah = 3;
	inregs.h.al = 'a';		/* ここに送信するデータを指定する */
	int86(0x19, &inregs, &outregs);
	if(outregs.h.ah) {
		printf("データ送信が失敗しました\n");
	}
}
```

5 データ受信

割り込み　INT 19H

入　力　AH←04H

出　力　AH→00H：正常終了

01H：RS-232Cが初期化されていない

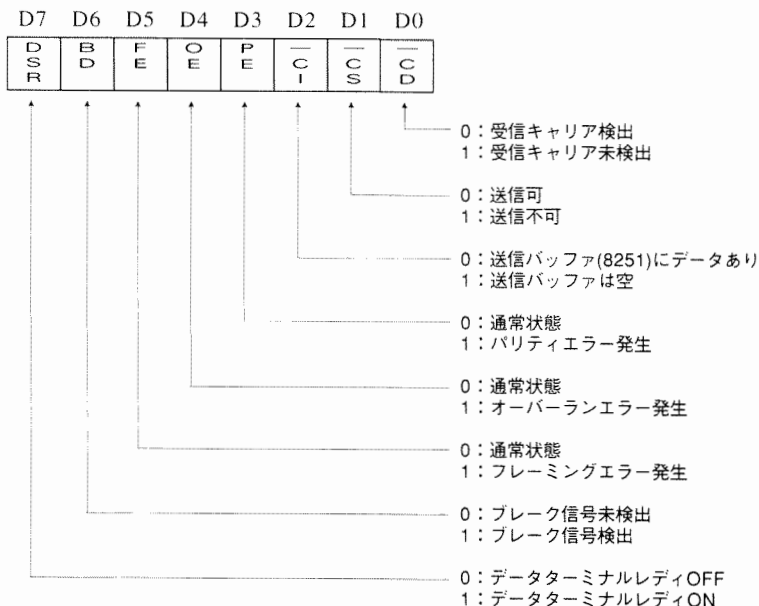
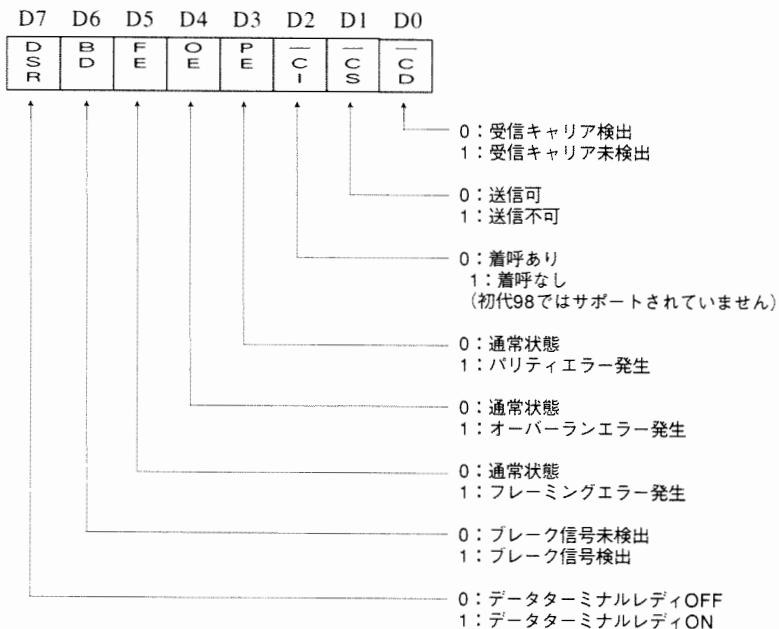
02H：受信バッファがオーバーフローしている

03H：8251が送信可，受信可の状態でない．

CH→受信データ

CL→受信データの受信時ステータス

解　説　受信バッファ内の受信データおよび，受信時ステータスを読みだします．受信時ステータスは，以下のようになっています．

・ノーマルモード時のステータス

・ハイレゾモード時のステータス

各エラーおよびブレーク信号の意味は，「2-10-1.RS-232CのI/Oポート」の「ステータス」を参照してください（p.307）．

このファンクションを呼び出したときに受信バッファが空の場合は，初期化コマンドで指定した受信時タイムアウト時間だけ待ち，その間にデータが送られてこなければ，エラーとなります．

サンプル

データを1バイト受信し，表示します．ただし，このプログラムを実行するためには，RS-232C(BIOS)が初期化されていなければなりません．

```
#include <stdio.h>
#include <dos.h>

union REGS inregs, outregs;

void main(void)
{
	inregs.h.ah = 4;
	int86(0x19, &inregs, &outregs);
	printf("受信データ=%d\n", outregs.h.ch);
}
```

6 8251へのコマンド出力

割り込み INT 19H

入　力 AH←05H

CL←8251へ出力するコマンド

出　力 AH→00H：正常終了

01H：RS-232Cが初期化されていない

02H：受信バッファがオーバーフローしている

解　説

RS-232CのLSI，8251へコマンドを出力します．CLレジスタに設定する値は，AH=00Hファンクション（初期化）のコマンド設定データと各ビットの意味は同じですので，そちらを参照してください．

なお，IRビットを1に設定すると，232Cをクローズするとみなして初期化していない状態にします．また，RXEビットを0にすると，RxRDY割り込みをマスクします．

7 ステータスの取得

割り込み INT 19H

入　力 AH←06H

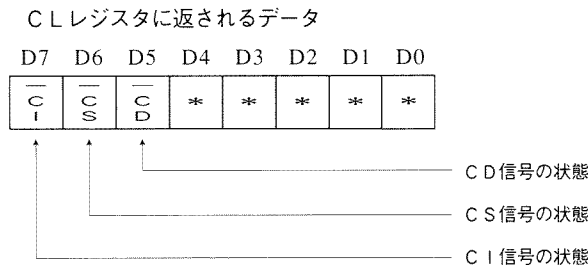
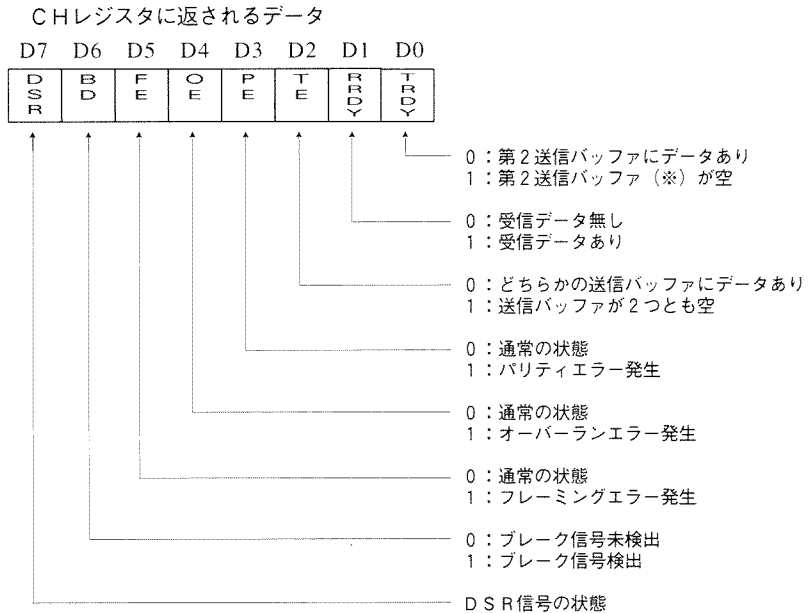
出　力 **AH**→00H：正常終了

01H：RS-232Cが初期化されていない

02H：受信バッファがオーバーフローしている

CH→8251のステータス

CL→モデムステータス

解　説 232Cの各信号のステータスを得ます．CHレジスタに返される値の詳細は，以下のようになっています．

※（初代98ではサポートされていません）

各エラーおよびブレーク信号の意味は，「2-10-1.RS-232CのI/Oポート」の「ステータス」を参照してください（p.307）．

8 RS-232C BIOS初期化（拡張モード）

割り込み INT 19H

入力 AH←07H
BX←BIOS制御情報
CH←8251モード設定データ
CL←8251コマンド設定データ
ES：DI←受信バッファの先頭番地
DX←受信バッファのサイズ（バイト単位）

出力 AH→00H：正常終了
04H：拡張RS-232Cボードが存在しない

解説 マルチタスク対応の動作モードでRS-232Cを初期化します．なお，このファンクションは，ハイレゾモードのみサポートされています．このファンクションを利用してRS-232Cを初期化すると，すべてのチャネル（拡張ポートを含めて3ch）からの受信割り込みを１つの割り込みチャネルで受け，その後，INT 1FHの内部割り込みが発生します．そのときに，AHレジスタには83H，ALレジスタにはどのRS-232Cのチャネルからの割り込みかを示すデータが渡されます．ALレジスタと，チャネルの関係は，表2-38に示すようになっています．

表2-38 ALとチャネル番号

ＡＬレジスタ	チャネル番号
１０Ｈ	1
１１Ｈ	2
１２Ｈ	3

BXレジスタに設定する値の各ビットの詳細は，以下のようになっています．

・BHレジスタ

・ＢＬレジスタ

D7	D6	D5	D4	D3	D2	D1	D0
*	*	*	*	*	*	*	BUFCTL

D0（BUFCTL）
0：受信バッファをバイト単位に管理する
1：受信バッファをワード単位に管理する

AH，BXレジスタ以外のレジスタに設定する値は，AH＝00HのRS-232C BIOSの初期化と同じですので，そちらを参照してください．

注）拡張モードでは，受信バッファがオーバーフローした場合，8251を初期化しなくてはいけません．

●拡張RS-232Cポートについて

ここでは，基本的に内蔵ポートのBIOSについて説明してきましたが，BIOSは拡張ポートにも対応しています．しかしながら，BIOSが対応しているのは，NEC純正の増設RS-232Cボードのみです．また，純正のボードは，ソフトウェアから通信速度の変更ができないということや，内蔵ポート同様，最高の通信速度が9600bpsであるという欠点があります．

以上のことに加え，安価で機能の高い多くの拡張RS-232Cボードが周辺機器メーカーから発売されていますので，純正の拡張RS-232Cボードは，利用されないことが多いといえます．しかもRS-232Cの拡張ボードに関しては，メーカー間の互換性は，まったくありません．よって，拡張ポートのBIOSを利用することは，めったにないといえます．

したがって，拡張RS-232CのBIOSについての説明は，簡単に触れる程度にします．

●ノーマルモードでのRS-232Cのチャネル指定

ノーマルモードでは，割り込み番号でチャネルを指定します．すべてのRS-232CのBIOSで，内蔵ポートではINT19Hですが，2ポート目はINT 0D4 H，3ポート目はINT 0D5Hとなります．

●ハイレゾモードでのRS-232Cのチャネル指定

ハイレゾモードでは，すべてのRS-232CBIOSにおいて，AHレジスタの上位4ビットでチャネルを指定します．詳しくは，表2-39のようになっています．

表2-39　AHの上位4ビットとチャネル

AH				チャネル番
D7	D6	D5	D4	
0	0	0	0	1
0	0	0	1	2
0	0	1	0	3
そのほか				無効

§ 2-11 マウス

最近では，マウスはキーボードとならぶ標準的な入力装置として，Windowsをはじめとする多くのアプリケーションによって利用されるようになりました．

一般的なマウスは本体底部の球状のローラーにより移動量を検出し，そのデータとマウス本体のボタン状態を本体側に送ります．これらの制御用LSIとして8255A（相当品）が使用されています．

PC-9801はマウス用のインターフェースを標準で内蔵しています（ただしPC-9801／E／Fではオプション）．

■2-11-1 ――― マウスのI/Oポート

マウスに関する制御はI/Oポートを直接操作するよりマウスドライバを利用する方法が簡単でかつ安全です．しかし，マウスI/Oの一部には98本体の各種切り替えスイッチやディップスイッチに関する情報が含まれていますので，これらを取得する場合はこのI/Oポートから読み込むことになります．また，マウスに関してもノーマルモードにおける割り込み周期はマウスドライバでは設定できませんので，変更する場合にはこのI/Oを直接制御する形になります．

■マウスI/O一覧（ノーマルモード）

リード／ライト	I／O アドレス	命令（機能）	データ D7	D6	D5	D4	D3	D2	D1	D0
リード	7FD9H	リードポートA（マウス状態取得）	LEFT	×	RIGHT	×	MD3	MD2	MD1	MD0
	7FDBH	リードポートB（ディップスイッチ取得1）	×	RAMKL	×	×	×	×	SPDSW	×
	7FDDH	リードポートC（ディップスイッチ取得2）	HC	SXY	SHL	-INT	MODSW	CPUSW	SW1-6	SW1-5

■マウスI/O一覧（ノーマルモード）（つづき）

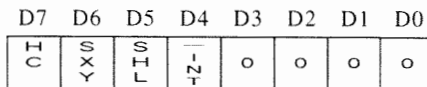
リード／ライト	I／O アドレス	命令（機能）	D7	D6	D5	D4	D3	D2	D1	D0
ライト	7FDFH	ライトモード	1	0	0	1	0	0	1	1
	7FDFH	割り込み　0:Enable　1:Disable	0	0	0	0	1	0	0	0/1
	7FDFH	カウントクリア 1:Clear	0	0	0	0	1	1	1	0/1
	7FDDH	ライトポートC	HC	SXY	SHL	－INT	0	0	0	0
	BFDBH	ライトタイマ	0	0	0	0	0	0	T1	T0

■マウスI/O一覧（ハイレゾモード）

リード／ライト	I／O アドレス	命令（機能）	D7	D6	D5	D4	D3	D2	D1	D0
リード	0061H	リードポートA（マウス状態取得）	LEFT	×	RIGHT	×	MD3	MD2	MD1	MD0
	0063H	リードポートB（ディップスイッチ取得1）	FDU	SW3｜6	1	1	1	1	1	MOD81
	0065H	リードポートC（ディップスイッチ取得2）	HC	SXY	SHL	－INT	MODSW	0	RS｜2	RS｜1
ライト	0067H	ライトモード	1	0	0	1	0	0	1	1
	0067H	割り込み　0:Enable　1:Disable	0	0	0	0	1	0	0	0/1
	0067H	カウントクリア　1:Clear	0	0	0	0	1	1	1	0/1
	0065H	ライトポートC	HC	SXY	SHL	－INT	0	0	0	0

1 ライトポートC/マウスI/Oポート

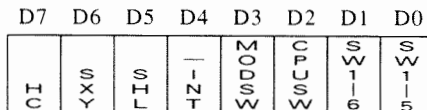
ノーマル 7FDDH

ハイレゾ 0065H

マウス割り込み許可とHCは表中のポートアドレス7FDFHの命令でも可能

サンプル マウス割り込みを禁止します.

```
unsigned char port_c=0x10;   /* 0x10=00010000b */
outportb(0x7fdd,port_c);
```

2 リードポートC/マウスI/Oポート ノーマル

ノーマル 7FDDH

※これらの読み出しはライトモードでモードを93Hに設定した後に行ってください.

サンプル　ポートCの状態を読み込みます．

```
outportb(0x7fdf,0x93);          /*ライトモード  */
port_c  =  inportb(0x7fdd);     /*リードポートC*/
```

port_cのビット0～3に各状態が読み込まれます．

2' リードポートC/マウスI/Oポート　ハイレゾ

ハイレゾ　**0065H**

3 リードポートB/マウスI/Oポート　ノーマル

ノーマル　**7FDBH**

サンプル　ポートBの状態を読み込みます．

```
port_b  =  inportb(0x7fdb);          /*リードポートB*/
```

port_bのビット1，6に各状態が読み込まれます．

3' リードポートB/マウスI/Oポート　ハイレゾ

ハイレゾ　0063H

4 リードポートA/マウスI/Oポート

ノーマル　7FD9H

ハイレゾ　0061H

D7	D6	D5	D4	D3	D2	D1	D0
LEFT	×	RIGHT	×	MD3	MD2	MD1	MD0

MD3〜MD0：SXY、SHLで指定されたカウントデータ

RIGHT：右側のマウスボタン状態
　0：押されている
　1：開放されている

LEFT：左側のマウスボタン状態
　0：押されている
　1：開放されている

5 ライトタイマ /マウスI/Oポート

ノーマル　BFDBH

割り込み周期設定

T1	T0	周期(ms)	周波数(Hz)
0	0	8.3	120
0	1	16.7	60
1	0	33.3	30
1	1	66.7	15

サンプル　マウスの割り込み周期を120Hzに設定します.

```
unsigned char bit_data=0x00;
outportb(0xbfdb,bit_data);        /*割り込み周期の設定*/
```

■サンプルプログラム

●ディップスイッチ状態の取得

いくつかのディップスイッチの設定状態を読み込んで画面に出力します. 機種によってはこのほかにも各種のモードスイッチが読み込めます.

```
#include<stdio.h>
#include<dos.h>

main()
{
      unsigned char port_b,port_c;
      unsigned char sw1-5,sw1-6,sw3-6,sw3-8;    /*各スイッチ状態の格納用*/
      char res_sw1[2][4]={"off","on"};          /*出力文字列設定*/
      char res_sw3[2][4]={"on","off"};

      outportb(0x7fdf,0x93);        /*ライトモード */
      port_c = inportb(0x7fdd);     /*リードポートC*/
      port_b = inportb(0x7fdb);     /*リードポートB*/

      sw1-5 = port_c & 0x01;              /*各スイッチに関係するビット情報のみを*/
      sw1-6 = port_c & 0x02) >> 1;        /*取り出す                          */
      sw3-6 = (port_b & 0x40) >> 6;
      sw3-8 = (port_c & 0x04) >> 2;

      printf("SW1-5(RS232C)           :%s\n"
             "SW1-6(RS232C)           :%s\n"
             "SW3-6(RAM)              :%s\n"
             "SW3-8(CPU Switch)       :%s\n"
         ,res_sw1[sw1-5],res_sw1[sw1-6],res_sw3[sw3-6],res_sw3[sw3-8]);

      return(0);
}
```

●マウス割り込み周期設定

マウス割り込みの周期の設定とそれによるマウスの動きの違いを確認するプログラムです. マウスドライバを使用しているので実行する際にはMOUSE.SYSを組み込んでください. 実行するとカーソルが表示されます. ここで右ボタンをクリックすると周期が8ms, 67msと切り替わり, 左ボタンをクリックすると終了します.

周期が67msの時には8msの時に比べると, かなり動きがぎこちなくなりますが, マウスによる割り込み回数が減るというメリットもうまれます.

```
#include<stdio.h>
#include<dos.h>
#include<stdlib.h>
```

```
main()
{
      int bit_data = 0; /* ポート出力データ */
      union REGS regs;

/*マウスドライバのチェックおよび初期化*/
      regs.x.ax = 0;                   /*機能コード00H*/
      int86(0x33,&regs,&regs);
      if (regs.x.ax == 0){
            puts("マウスが使えなーい");
            exit(1);
      }
/*マウスカーソルの表示*/
      regs.x.ax = 1;                   /*機能コード01H*/
      int86(0x33,&regs,&regs);

      while(1){

            if(bit_data == 0){
                  bit_data = 3;      /*周期67ms (T1=1, T0=1) */
                  printf("割り込み周期: 67 ms \r");
            } else {
                  bit_data = 0;      /*周期8ms (T1=0, T0=0) */
                  printf("割り込み周期:  8 ms \r");
            }
            outportb(0xbfdb,bit_data);       /*割り込み周期の設定*/

            do{                                        /*ボタン入力待ち* /
                  regs.x.ax = 3;
                  int86(0x33,&regs,&regs);
            } while (regs.x.bx == 0 && regs.x.ax == 0);
            if(regs.x.bx) { break; }

            while (regs.x.ax != 0){           /*ボタンの開放待ち*/
                  regs.x.ax = 3;
                  int86(0x33,&regs,&regs);
            }
      }

/*マウスカーソル消去*/
      regs.x.ax = 2;                   /*機能コード02H*/
      int86(0x33,&regs,&regs);

      return(0);
}
```

●マウスカウンタのデータの取得（マウス移動量の検出）

マウス移動量の検出など，マウスに関する制御は冒頭でも述べたとおりマウスドライバを使うべきです．しかし，マウスドライバが使用できない環境であったり，割り込みの関係上使用が困難な場合は直接これらをIOポートから取得せねばなりません．

次のサンプルはIO直接制御によるマウス移動量の検出をおこなうプログラムです．
マウスカウンタのデータ幅は8ビットですのでカウンタ値は256ずつループします．

```
#include<stdio.h>
#include<dos.h>

main(void)
{
      unsigned char xh,xl,yh,yl,x,y;

      outportb(0x7fdd,0x80);    /*カウンタクリア*/

      do{
            outportb(0x7fdd,0x00);         /*ポートAへの出力データ指定（Xの下位4ビット）*/
            xl = inportb(0x7fd9) & 0x0f;/*ポートAよりデータの取得    */
            outportb(0x7fdd,0x20);         /*ポートAへの出力データ指定（Xの上位4ビット）*/
            xh = inportb(0x7fd9) & 0x0f;/*ポートAよりデータの取得    */
            outportb(0x7fdd,0x40);         /*ポートAへの出力データ指定（Yの下位4ビット）*/
            yl = inportb(0x7fd9) & 0x0f;/*ポートAよりデータの取得    */
            outportb(0x7fdd,0x60);         /*ポートAへの出力データ指定（Yの上位4ビット）*/
            yh = inportb(0x7fd9) & 0x0f;/*ポートAよりデータの取得    */

             x = (xh<<4)+xl;               /*上位4ビットと下位4ビットの結合*/
            y = (yh<<4)+yl;

            printf("X=%3d   Y=%3d\r",x,y);

      }while( inportb(0x7fd9) & 0x80 );        /*マウスの左ボタンチェック*/

      return(0);
}
```

■2-11-2 ―――――― マウスBIOS（ドライバ）の種類

初期の98ではマウスは標準ではサポートされておらず，そのためにマウスBIOSはソフトウェアドライバとして提供されています．ただしハイレゾモードではマウス制御の機能をROM BIOS内で利用できるため，ソフトウェアドライバは必要としません．

さらにソフトウェアドライバにはNEC製のMS-DOSに付属しているMOUSE.SYSとマイクロソフト仕様のドライバ（一般にはMOUSE.COMの形で提供されている）が存在し，これがユーザーの環境を作る上で問題を複雑にさせています．

現在，市販のMS-DOS用アプリケーションではマウスの制御をI/Oポートにより直接行うもの（または自社独自のドライバを利用）が多く，フリーウェア※1などではどちらかのソフトウェアドライバが利用されているようです（最近ではマイクロソフト仕様を使用するのものが多いようです）．そこで本書ではこの2つのドライバとハイレゾ時のROM BIOSの3つについてそれぞれ別々に説明します．

なお，多くのC言語処理系ではこのマウスBIOS（ドライバ）機能をそのまま関数として利用できるようになっているため，それらを使用してマウスの制御を行うこともできます．

※1コンピュータ通信上などで自由に手に入れられ，利用することができるソフトウェア

●NEC仕様とマイクロソフト仕様ドライバの違い

これらが提供する機能コードのうち，多くの機能は共通の手続きにより同様に利用することができます．しかし，決定的な違いはNEC仕様のドライバはマウスカーソルをG-VRAMの1プレーンに対してのみ表示しますが，マイクロソフト仕様ドライバは4プレーンに対して表示できる点です．この違いは，背景の画面に何らかの絵，グラフなどが表示されているような場合，次のような現象となってあらわれます．NEC仕様のドライバは1プレーンに対してのみ表示されるので，カーソル位置のほかのプレーンの状態によりカーソルの表示色が変化します．しかしマイクロソフト仕様のドライバは4プレーンに対して表示できるため表示色を変化させないようにできます．このようなNECドライバの仕様は初期の98における処理能力などを考慮にいれたものと思われますが，最近の98の処理能力を考えると4プレーン表示でも特に問題ないと考えられます．

なお，ハイレゾ時のROM BIOSにおける表示は4プレーンに対し，かなり自由に表示できます．

◆各ドライバの準備

・NEC仕様のドライバを用いる場合

CONFIG.SYSファイルにMS-DOS添付のMOUSE.SYSを組み込む．※2
（またはADDDRV等のコマンドを利用して組み込む）

・マイクロソフト仕様のドライバを用いる場合

MOUSE.COM（マイクロソフトの製品，またはMS-DOS ver.5.0 などに付属）を実行するか，フリーウェアなどで手に入れられる互換ドライバ等を組み込む．

・ハイレゾモード時

ROM BIOSなのでドライバの組み込みは必要ないが，次のように初期設定を行う必要がある．

①使用されていない適当な割り込みベクタにマウス制御機能のエントリポイント，F800H：7FD0Hをセットする．
②機能コード[11H]（コマンド説明参照）によりINT2の割り込みに対するベクタテーブルの設定，割り込みビットマスクの解除を行う（マウスBIOSのワークエリアとして1916バイト以上を確保しておく必要がある）．

また，ハイレゾモード時でもソフトウェアドライバを組み込むことにより，ノーマルモードと同等の機能コードを使用することもできます．

※2起動ディスクのルートに存在するCONFIG.SYSファイルにエディタ等を用いて“DEVICE＝A：¥MOUSE.SYS”の一行加えてください（MOUSE.SYSが存在するドライブやディレクトリがこの例と異なる場合，“A:¥”の部分を変更してください）．

■マウスBIOS（NEC仕様）一覧（INT 33H）

コード	機　　　能	マイクロソフト仕様相違
００Ｈ	マウスＢＩＯＳ初期化	初期化項目
０１Ｈ	マウスカーソル表示	
０２Ｈ	マウスカーソル消去	呼出回数のスタックあり
０３Ｈ	マウスカーソル位置，ボタン状態取得	ボタンの情報形式
０４Ｈ	マウスカーソル位置設定	
０５Ｈ	左ボタン押下情報取得	左右ボタン指定
０６Ｈ	左ボタン開放情報取得	左右ボタン指定
０７Ｈ	右ボタン押下情報取得	（機能コード０５Ｈ）
０８Ｈ	右ボタン開放情報取得	（機能コード０６Ｈ）
０９Ｈ	マウスカーソル形状設定	形状データ形式
０ＢＨ	マウスカーソルの移動距離取得	
０ＣＨ	ユーザー定義サブルーチンのコール条件設定	
０ＦＨ	ミッキー／ドット比設定	
１０Ｈ	水平方向のマウスカーソル移動範囲設定	機能コード０７Ｈ
１１Ｈ	垂直方向のマウスカーソル移動範囲設定	機能コード０８Ｈ
１２Ｈ	カーソル表示画面設定	機能コードＦＦＨ
１３Ｈ	グラフィックＶＲＡＭ設定と実装状況取得	機能コードＦＤＨ
１４Ｈ	マウス割り込みの許可	機能コードＦＥＨ

1 マウスBIOS初期化

割り込み INT　33H

入　　力 AH←00H

出　　力 AX→マウスの環境状態　0：使用不可能
　　　　　　　　　　　　　　　　-1：使用可能

解　　説 マウスが使用可能であるかどうかをチェックし，使用可能であればマウス環境を初期化します．ドライバ組み込み時の初期状態を以下に示しますが，このコマンドにより*のついた項目が再初期化されます（*がついていない項目はこのコマンドでは初期化さ

れない).

ドライバ組み込み時の初期状態

カーソル表示*	表示しない
カーソル位置	(319, 399) 400ライン時 (319, 199) 200ライン時
カーソル表示画面	第2プレーン
カーソル移動範囲	画面全体 (0, 0) ~ (639, 199) 400ライン時 (0, 0) ~ (639, 99) 200ライン時
カーソル形状*	左上向きの大きな矢印 (16×32ドット) 400ライン時 (16×16ドット) 200ライン時
カーソル中心点*	(0, 0)
ミッキー/ドット比*	水平方向8, 垂直方向8
ユーザー定義サブルーチンのコール条件*	0
マウス割り込み周期	8ms

サンプル マウスBIOSのチェックと初期化を行います.

```
union  REGS  regs;

regs.x.ax  =  0x00;                           /*機能コード00H*/
int86(0x33,&regs,&regs);
if  (regs.x.ax  ==  0){
        puts("マウスが使えなーい");
        exit(1);
}  else  {
        puts("マウスドライバを初期化したぞ");
}
```

2 マウスカーソル表示 NEC

割り込み INT 33H

入　力 AX←01H

出　力 なし

解　説 マウスカーソルを画面上に表示します. このコマンドを実行すると, カーソルの消去またはマウスBIOS初期化が実行されるまで表示し続けます.

サンプル マウスカーソルを表示します.

```
union REGS regs;

regs.x.ax =0x01;                  /*機能コード01H*/
int86(0x33,&regs,&regs);
```

3 マウスカーソル消去 NEC

割り込み INT 33H

入　力 AX←02H

出　力 なし

解　説 マウスカーソルを画面上より消去します. このコマンドを実行すると, マウスカーソルの表示が再び実行されるまでカーソルは表示されません. ただし, 表示されていない間もマウス本体の移動にあわせてのカーソルは画面上を移動しています. したがって再びカーソルを表示させたときは非表示の間に移動したカーソル位置に表示されます.

サンプル マウスカーソルを消去します.

```
union REGS regs;

regs.x.ax = 0x02;                 /*機能コード02H*/
int86(0x33,&regs,&regs);
```

4 マウスカーソル位置, ボタン状態取得 NEC

割り込み INT 33H

入　力 AX←03H

出　力 AX→左ボタン状態　0：開放されている
　　　　　　　　　　　　　　−1：押されている

BX→右ボタン状態　0：開放されている
　　　　　　　　　−1：押されている

CX→カーソル位置の水平座標　0〜639

DX→カーソル位置の垂直座標　0〜399：400ライン時
　　　　　　　　　　　　　　0〜199：200ライン時

解　説 マウスカーソルの位置とボタンの状態を取得します. このコマンドによりカーソルの水平, 垂直の両座標, 並びに左右ボタンの状態をすべて一度に得ることができます.

サンプル　マウスカーソル位置，ボタン状態の取得を取得し，その結果を画面に表示します．左ボタンを押すと終了します．

```
union  REGS  regs;

do{
        regs.x.ax  =  0x03;                /*機能コード03H*/
        int86(0x33,&regs,&regs);
      printf("x  =  %3d,  y  =  %3d,左ボタン  =  %2d,右ボタン  =  %2d  \r"
                        ,regs.x.cx,regs.x.dx,regs.x.ax,regs.x.bx);
}  while  (regs.x.ax  ==  0  );     /*左ボタンを押すと止まります*/
```

5　マウスカーソル位置設定　NEC

割り込み　INT　33H

入　　力　AX←04H

CX←新たに設定するカーソル位置の水平座標

DX←新たに設定するカーソル位置の垂直座標

出　　力　なし

解　　説　マウスカーソルの位置を新たに任意の位置に移動します．設定できる座標は画面のモードと機能コード11H（カーソル移動範囲の設定）により設定された範囲内で決まります．また，この範囲を超えた場合，移動範囲内の端にカーソルは移動します．

サンプル　カーソルをディスプレイの中心に設定します．

```
union  REGS  regs;

regs.x.ax  =  0x04;              /*機能コード04H*/
regs.x.cx  =  320;               /*カーソルX座標指定*/
regs.x.dx  =  200;               /*カーソルY座標指定*/
int86(0x33,&regs,&regs);
```

6　左ボタン押下情報取得　

割り込み　INT　33H

入　　力　AX←05H

出　　力　AX→左ボタン状態　　0：開放されている

　　　　　　　　　　　　　　　−1：押されている

BX→前回呼び出し後からの押された回数

CX→最後に押されたときのカーソル位置の水平座標

DX→最後に押されたときのカーソル位置の垂直座標

解　説　左ボタンが押されたときに関する各種情報を取得します．ボタンの状態は機能コード03H（マウスカーソル位置，ボタン状態取得）でも得られますが，このコマンドではさらに，最後にボタンが押されたときのカーソルの座標およびこのコマンドが前回呼び出された後から左ボタンが押された回数を取得することができます．したがって，このコマンドを用いれば常時マウスの状態を追いかけることなくボタンを押したときのカーソルの座標を読み取ることができます．

サンプル　p.353のサンプルを参考にしてください．

7　左ボタン開放情報取得　NEC

割り込み　INT　33H

入　力　AX←06H

出　力　AX→左ボタン状態　0：開放されている

−1：押されている

BX→前回呼び出し後からの離された回数

CX→最後に離されたときのカーソル位置の水平座標

DX→最後に離されたときのカーソル位置の垂直座標

解　説　左ボタンが離されたときに関する各種情報を取得します．ボタンの状態は機能コード03H（マウスカーソル位置，ボタン状態取得）でも得られますが，このコマンドではさらに，最後にボタンが離されたときのカーソルの座標およびこのコマンドが前回呼び出された後から左ボタンが離された回数を取得することができます．このコマンドは機能コード05H（左ボタン押下状態取得）において，ボタンを〈押されたときの状態〉が，〈離されたときの状態〉に，変わったものと考えられます．

サンプル　p.353のサンプルを参考にしてください．

8　右ボタン押下情報取得　

割り込み　INT　33H

入　力　AX←07H

出力 AX→右ボタン状態 0：開放されている
−1：押されている

BX→前回呼び出し後からの押された回数

CX→最後に押されたときのカーソル位置の水平座標

DX→最後に押されたときのカーソル位置の垂直座標

解説 右ボタンが押されたときに関する各種情報を取得します．ボタンの状態は機能コード03H（マウスカーソル位置，ボタン状態取得）でも得られますが，このコマンドではさらに，最後にボタンが押されたときのカーソルの座標およびこのコマンドが前回呼び出された後から右ボタンが押された回数を取得することができます．したがって，このコマンドを用いれば常時マウスの状態を追いかけることなくボタンを押したときのカーソルの座標を読み取ることができます．

サンプル p.353のサンプルを参考にしてください．

9 右ボタン開放情報取得 NEC

割り込み INT 33H

入力 AX←08H

出力 AX→右ボタン状態 0：開放されている
−1：押されている

BX→前回呼び出し後からの離された回数

CX→最後に離されたときのカーソル位置の水平座標

DX→最後に離されたときのカーソル位置の垂直座標

解説 右ボタンが離されたときに関する各種情報を取得します．ボタンの状態は機能コード03H（マウスカーソル位置，ボタン状態取得）でも得られますが，このコマンドではさらに，最後にボタンが離されたときのカーソルの座標およびこのコマンドが前回呼び出された後から右ボタンが離された回数を取得することができます．このコマンドは機能コード07H（右ボタン押下状態取得）において，ボタンを＜押されたときの状態＞が＜離されたときの状態＞に変わったものと考えられます．

サンプル p.353のサンプルを参考にしてください．

10 マウスカーソル形状設定 NEC

割り込み INT　33H

入　力 AX←09H

BX←カーソルの中心点の水平座標　0～15

CX←カーソルの中心点の垂直座標　0～31：400ライン時

0～15：200ライン時

ES：DX←カーソル形状データの先頭アドレス

データ形式は16×32ビット：400ライン時

16×16ビット：200ライン時

出　力 なし

解　説

マウスカーソルの形状および中心点を設定します．マウスカーソルは16×32ドット（400ライン時）または16×16ドット（200ライン時）のデータで形成されており，中心点とはそのカーソルデータ内のどのドットがマウスの指し示す座標となるかを決定するものです．中心点はカーソル形状データの左上を（0，0），右下を（15，31または15）とした時，この範囲内で自由に設定することができます．初期状態ではカーソル形状は左上向きの矢印となっていますから矢印の先端（0，0）に中心点が設定されています（図2-38）．

マウスカーソルの形状データのフォーマットはカーソル形状データの座標で（0，0），（1，0），……，（15，0），（0，1），（2，1），……，（15，32または15）の順に1ビットずつ（各ドット1の時，表示）並べたもので全部で64バイト（400ライン時），32バイト（200ライン時）となります．

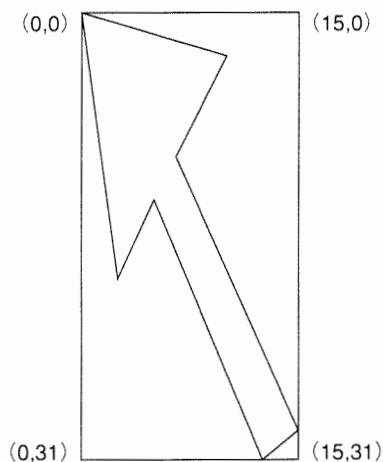



図2-38マウスデータ形式（NEC）

サンプル カーソルの形状を上向きの小さな矢印に変更します.

```
char  keijou[2*32]={
      0x01,0x00,              /*0000000100000000*/
      0x03,0x80,              /*0000001110000000*/
      0x07,0xc0,              /*0000011111000000*/
      0x0f,0xe0,              /*0000111111100000*/
      0x1f,0xf0,              /*0001111111110000*/
      0x3f,0xf8,              /*0011111111111000*/
      0x7f,0xfc,              /*0111111111111100*/
      0xff,0xfe,              /*1111111111111110*/
      0x07,0xc0,              /*0000011111000000*/
      0x07,0xc0,              /*0000011111000000*/
      0x07,0xc0,              /*0000011111000000*/
      0x07,0xc0,              /*0000011111000000*/
      0x07,0xc0,              /*0000011111000000*/
      0x07,0xc0,              /*0000011111000000*/
      0x07,0xc0,              /*0000011111000000*/
      0x07,0xc0,              /*0000011111000000*/
      0x00,0x00,
      0x00,0x00,
      0x00,0x00,
      0x00,0x00,
      0x00,0x00,
      0x00,0x00,
      0x00,0x00,
      0x00,0x00,
      0x00,0x00,
      0x00,0x00,
      0x00,0x00,
      0x00,0x00,
      0x00,0x00,
      0x00,0x00,
      0x00,0x00,
      0x00,0x00,
};

union  REGS  regs;

regs.x.ax  =  0x09;                  /*機能コード09H*/
regs.x.bx  =  7;                     /*カーソル中心点の水平座標*/
regs.x.cx  =  0;                     /*カーソル中心点の垂直座標*/
regs.x.dx  =  (int)keijou;     /*カーソル形状データのアドレス設定*/
int86(0x33,&regs,&regs);
```

11 マウスカーソルの移動距離取得 NEC

割り込み INT　33H

入　力 AX←0BH

出　力 CX→マウスの水平方向移動距離　−32768〜32767[ミッキー]

DX→マウスの垂直方向移動距離　−32768〜32767[ミッキー]

解　　説　マウスの移動距離を取得します．このコマンドが前回実行された時から今回実行される時までのマウス位置の相対的な移動距離がミッキー単位で出力されます．ここで水平方向は右向き，垂直は下向きが正の向きとなります．

サンプル　前回左ボタンが押された時から今回左ボタンを押した時までのマウスの移動量を画面に表示します．右ボタンで終了します．

```
union REGS regs;

while(1){
        do{                                                /*ボタンの入力待ち*/
                regs.x.ax = 0x03;
                int86(0x33,&regs,&regs);
        } while (!(regs.x.ax || regs.x.bx));
        if(regs.x.bx){ break; }

        regs.x.ax = 0x0b;                         /*機能コード0BH*/
        int86(0x33,&regs,&regs);
        printf("x = %4d , y = %4d \r",regs.x.cx,regs.x.dx);

        do{                                                /*ボタンの開放待ち*/
                regs.x.ax = 0x03;
                int86(0x33,&regs,&regs);
                } while (regs.x.ax);
}
```

12 ユーザー定義サブルーチンのコール条件設定 NEC

割り込み　**INT　33H**

入　　力　**AX**←**0CH**

CX←コール条件

ビット0：マウスカーソルの位置変化
ビット1：左ボタンの押下
ビット2：左ボタンの開放
ビット3：右ボタンの押下
ビット4：右ボタンの開放
各ビット1のときにコールする
複数指定可能

ES：DX←ユーザー定義サブルーチンのアドレス

出　　力　なし

解　　説　ユーザーが作成したサブルーチンをマウスドライバがコールする条件とそのアドレス

を設定します．このコールは次の手順で行われます．まずマウス割り込みによって制御がマウスドライバに移ります．次に，マウスドライバは指定されたコール条件のうち1つでも条件を満たせばサブルーチンをコールします．ここで，マウスドライバからのコールはユーザー定義サブルーチンがFAR型プロシージャとして行われます．したがってユーザーはサブルーチンをFAR型のプロシージャとして作成してください．

また，マウスドライバがサブルーチンをコールする際には次のような情報をレジスタに格納してコールします．よって，ユーザー定義サブルーチン内ではこれらのレジスタを参照することによりマウスの情報を利用できます．

AX→コールの原因となった現象
　1：カーソルの位置変化
　2：左ボタンの押下
　4：左ボタンの開放
　8：右ボタンの押下
　16：右ボタンの開放
BL→左ボタンの状態
　0：開放されている
　−1：押されている
BH→右ボタンの状態
　0：開放されている
　−1：押されている
CX→カーソル位置の水平座標
DX→カーソル位置の垂直座標

13　ミッキー／ドット比設定　

割り込み　INT　33H

入　力　AX←0FH
CX←水平方向のミッキー／ドット比
DX←垂直方向のミッキー／ドット比

出　力　なし

解　説　マウス本体の机上の移動距離（ミッキー単位）とそれに対応する画面上のマウスカーソルの移動距離（ドット単位）の比を設定します．この設定はマウスカーソルを8ドット移動させるのに要するマウス本体の机上の移動距離（ミッキー／8ドット，1ミッキーは約0.25mm）を単位として設定します．（水平，垂直方向はそれぞれ個別に設定可能）．したがってこの値を大きくすればマウスの感度は低くなり，値を小さくすればマウスの感度が高くなります．また，負の値を入れることにより，マウスの移動方向を反

転させることもできます．

サンプル　水平方向のミッキー／ドット比を4，垂直方向のミッキー／ドット比を16に設定します．

```
union  REGS  regs;     /*  dos.hをインクルードしておくこと  */

regs.x.ax  =  0x0f;     /*機能コード0FH（ミッキー／ドット比設定）*/
regs.x.cx  =   4;      /* 水平方向のミッキー／ドット比            */
regs.x.dx  =  16;      /* 水平方向のミッキー／ドット比            */
int86(0x33,&regs,&regs);
```

14 水平方向のマウスカーソル移動範囲設定 NEC

割り込み　**INT　33H**

入　力　**AX←10H**

CX←カーソルの水平方向の移動範囲の最小値　0～639

DX←カーソルの水平方向の移動範囲の最大値　0～639

出　力　なし

解　説　マウスカーソルの水平方向の画面上移動範囲を設定します．ここで設定された移動範囲がカーソルの中心点が移動できる範囲となります（p.346, 図2-39）．なお，CXレジスタの値がDXレジスタより大きかった場合CXが最大値，DXが最小値となります．

サンプル　カーソルの水平方向の移動範囲を100～540に設定します．

```
union  REGS  regs;

regs.x.ax  =  0x10;                    /*機能コード10H*/
regs.x.cx  =  100;
regs.x.dx  =  540;
int86(0x33,&regs,&regs);        /*X座標範囲の設定*/
```

15 垂直方向のマウスカーソル移動範囲設定 NEC

割り込み　**INT　33H**

入　力　**AX←11H**

CX←カーソルの垂直方向の移動範囲の最小値　0～399：400ライン時
0～199：200ライン時

DX←カーソルの垂直方向の移動範囲の最大値　0～399：400ライン時
0～199：200ライン時

出力 なし

解説 マウスカーソルの垂直方向の画面上移動範囲を設定します．ここで設定された移動範囲がカーソルの中心点が移動できる範囲となります（図2-39）．なお，CXレジスタの値がDXレジスタより大きかった場合CXが最大値，DXが最小値となります．

サンプル カーソルの垂直方向の移動範囲を50～350に設定します．

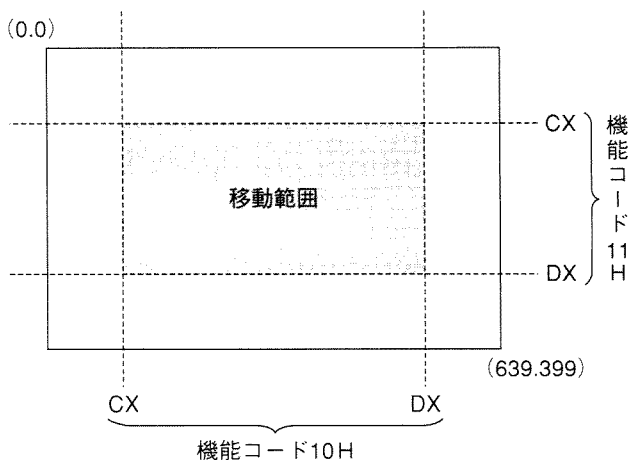
```
union  REGS  regs;

regs.x.ax  =  0x11;                    /*機能コード11H*/
regs.x.cx  =  50;
regs.x.dx  =  350;
int86(0x33,&regs,&regs);      /*Y座標範囲の設定*/
```


図2-39 マウスの移動範囲

16 カーソル表示画面設定 NEC

割り込み INT 33H

入力 AX←12H

BX←マウスカーソルの表示プレーン 0～3

出力 なし

解説 カーソルの表示画面を設定します．このコマンドの実行によりカーソルの色はそのプレーン（0～3を指定）に割り当てられた表示色となります．ただし，NEC仕様のマウスドライバのカーソル表示は1枚のプレーンに対してのみ処理が行われるので，各プレーンのドットがすべて非表示の状態以外（つまり背景が黒でなく絵や表などが表示されている場合），カーソル色は他のプレーンとの状態により設定されるパレット色となりま

す．したがって画面上の場所によってカーソルの色は変化してしまいます．

プレーン3は16色に対応していない機種（16色ボードが実装されているものは除く）には存在しないため，設定しても表示色は変わりません．また，16色に対応している機種でも機能コード13H（グラフィックVRAM設定と実装状況取得）で，プレーン3の使用を宣言しないと利用できませんので注意してください．

サンプル 表示画面をプレーン1に設定します．

```
union  REGS  regs;

regs.x.ax  =  0x12;                 /*機能コード12H*/
regs.x.bx  =  1;                    /*プレーンの指定              */
int86(0x33,&regs,&regs);
```

17 グラフィックVRAM設定と実装状況取得 NEC

割り込み INT　33H

入　力 AX←13H

BX←グラフィックVRAMの使用画面設定

0：プレーン0～2使用

1：プレーン0～3使用

出　力 BX→グラフィックVRAM実装状態

0：プレーン3を実装してない

−1：プレーン3を実装している

解　説 グラフィックVRAMの使用プレーン設定と実装状況の取得をします．プレーン3は16色に対応していない機種（16色ボードが実装されているものは除く）には存在しないため使用できません．また，16色対応機種でもプレーン3はこのコマンドで使用を宣言しないと利用できません．

サンプル プレーン3の実装チェックを行い，使用プレーンを0～3に設定します．

```
union  REGS  regs;

regs.x.ax  =  0x13;                 /*機能コード13H*/
regs.x.bx  =  1;
int86(0x33,&regs,&regs);
if  (regs.x.bx  ==  0){
        puts("プレーン3がないぞ");
}
```

18　マウス割り込みの許可

割り込み　INT　33H

入　　力　AX←14H

BX←マウス割り込み（8255Aの割り込み）制御

0：割り込み開始

1：割り込み停止

出　　力　なし

解　　説　マウス割り込みを停止，開始させます．

■サンプルプログラム

●カーソルの表示

まずは，機能コード0～03Hまでを使ったプログラムです．マウスドライバの組み込みを忘れないでください．

実行させるとマウスカーソルが表示されて画面内を自由に動かせます．左ボタンを押すことにより停止します．最後にカーソルの消去を行います．

```
#include<stdio.h>
#include<dos.h>
#include<stdlib.h>

main()
{
      union  REGS  regs;

/*マウスドライバのチェックおよび初期化*/
      regs.x.ax  =  0;                        /*機能コード00H*/
      int86(0x33,&regs,&regs);
      if  (regs.x.ax  ==  0){
            puts("マウスが使えなーい");
            exit(1);
      } else  {
            puts("マウスドライバを初期化したぞ");
      }

/*マウスカーソルの表示*/
      regs.x.ax  =  1;                        /*機能コード01H*/
      int86(0x33,&regs,&regs);

/*マウスカーソル位置，ボタン状態の取得*/
      do{
```

```
        regs.x.ax  =  3;                /*機能コード03H*/
        int86(0x33,&regs,&regs);
        printf("x  =  %3d,  y  =  %3d,左ボタン  =  %2d,右ボタン  =  %2d  \r"
                          ,regs.x.cx,regs.x.dx,regs.x.ax,regs.x.bx);
    } while  (regs.x.ax  ==  0  );
    /*左ボタンを押すと止まります*/

/*マウスカーソル消去*/
    regs.x.ax  =  2;                    /*機能コード02H*/
    int86(0x33,&regs,&regs);

    return(0);
}
```

●各種設定を行ったカーソルの表示

下の各項目について設定を行ったマウスカーソルを表示させます．プログラム中でどの機能をどのように使用しているかを確認してみてください．

1）カーソル形状変更

初期状態のカーソルは大きくてあまりカッコよくないので上向きの小さな矢印に変更しました．真上向きなので左利きの人も違和感なく使えると思います．みなさんも自分で気に入った形や，中心点に書き換えてみてください．

2）ミッキー／ドット比変更

変更をわかりやすくするために縦横の比を4倍にとりました．この設定では横方向には早く，縦方向には遅くカーソルが移動します．

3）移動範囲の設定

カーソル移動範囲を小さくします．なお，このプログラム実行後は移動範囲は常にこの範囲内となり，BIOS初期化（機能コード00H）を実行しても初期化されません．したがってもとに戻すときはこのコマンドで範囲を設定しなおす必要があります．

4）表示プレーンの指定

表示プレーンをプレーン1とします．また，int planeを変更して表示プレーンをいろいろ変えてみてください．アナログパレットが初期値であれば，0−青，1−赤，2−緑，3−灰でカーソルが表示されます．

```
#include<stdio.h>
#include<dos.h>
#include<stdlib.h>

char  keijou[2*32]={
      0x01,0x00,                 /*0000000100000000*/
      0x03,0x80,                 /*0000001110000000*/
      0x07,0xc0,                 /*0000011111000000*/
      0x0f,0xe0,                 /*0000111111100000*/
      0x1f,0xf0,                 /*0001111111110000*/
      0x3f,0xf8,                 /*0011111111111000*/
      0x7f,0xfc,                 /*0111111111111100*/
```

```
        0xff,0xfe,                      /*1111111111111110*/
        0x07,0xc0,                      /*0000011111000000*/
        0x07,0xc0,                      /*0000011111000000*/
        0x07,0xc0,                      /*0000011111000000*/
        0x07,0xc0,                      /*0000011111000000*/
        0x07,0xc0,                      /*0000011111000000*/
        0x07,0xc0,                      /*0000011111000000*/
        0x07,0xc0,                      /*0000011111000000*/
        0x07,0xc0,                      /*0000011111000000*/
        0x00,0x00,
        0x00,0x00,
        0x00,0x00,
        0x00,0x00,
        0x00,0x00,
        0x00,0x00,
        0x00,0x00,
        0x00,0x00,
        0x00,0x00,
        0x00,0x00,
        0x00,0x00,
        0x00,0x00,
        0x00,0x00,
        0x00,0x00,
        0x00,0x00,
        0x00,0x00,
};

main()
{
        union REGS regs;
        int x_ratio=4,y_ratio=16;               /*ミッキー/ドット比指定*/
        int xmin=100,xmax=540,ymin=50,ymax=350; /*移動範囲指定          */
        int plane=1;                            /*表示プレーン指定      */

/*マウスドライバのチェックおよび初期化*/
        regs.x.ax = 0x00;                       /*機能コード00H*/
        int86(0x33,&regs,&regs);
        if (regs.x.ax == 0){
                puts("マウスが使えなーい");
                exit(1);
        }

/*マウスカーソル形状設定*/
        regs.x.ax = 0x09;                       /*機能コード09H*/
        regs.x.bx = 7;                          /*カーソル中心点の水平座標*/
        regs.x.cx = 0;                          /*カーソル中心点の垂直座標*/
        regs.x.dx = (int)keijou;                /*カーソル形状データのアドレス設定*/
        int86(0x33,&regs,&regs);

/*ミッキー/ドット比設定*/
        regs.x.ax = 0x0f;                       /*機能コード0FH*/
        regs.x.cx = x_ratio;
        regs.x.dx = y_ratio;
        int86(0x33,&regs,&regs);
```

```
/*マウス移動範囲の設定*/
      regs.x.ax  =  0x10;                    /*機能コード10H*/
      regs.x.cx  =  xmin;
      regs.x.dx  =  xmax;
      int86(0x33,&regs,&regs);               /*X座標範囲の設定*/

      regs.x.ax  =  0x11;                    /*機能コード11H*/
      regs.x.cx  =  ymin;
      regs.x.dx  =  ymax;
      int86(0x33,&regs,&regs);               /*Y座標範囲の設定*/

/カーソル表示プレーンの設定*/
      if(plane == 3){
            regs.x.ax  =  0x13;              /*機能コード13H (GRAM4画面設定) */
            regs.x.bx  =  1;
            int86(0x33,&regs,&regs);
            if (regs.x.bx == 0){
                  puts("プレーン3がないぞ");
            }
      }
      regs.x.ax  =  0x12;                     /*機能コード12H (カーソル表示画面設定)
*/
      regs.x.bx  =  plane;
      int86(0x33,&regs,&regs);


/*マウスカーソルの表示*/
      regs.x.ax  =  1;                       /*機能コード01H*/
      int86(0x33,&regs,&regs);

/*左ボタンが押されるまでループ*/
      do{
            regs.x.ax  =  3;
            int86(0x33,&regs,&regs);
      } while (regs.x.ax == 0);

/*マウスカーソル消去*/
      regs.x.ax  =  0x02;                    /*機能コード02H*/
      int86(0x33,&regs,&regs);

      return(0);
}
```

●シングルクリック，ダブルクリックの検出

最後に簡単な応用例です．マウスの操作では1回ボタンをクリック（押して離すこと）するか，2回連続でクリックするかで，処理を分岐させることがよくあります．これらの処理は特別なBIOSコマンドが用意されているわけではなく，プログラム上で判断することになります．

まず，マウスのクリックの判断ですがマウスを押したことのみチェックしてもうまくいきません．これは，人がボタンを押してすぐ離したと思っても，押している状態のときにプログラム上ではすぐに次の押下チェックが行われ，結果的に数回押されたことになってしまうからです．したがって一度押されたら次に離されたことをチェックする必要があります（これはキーボードなどでもよく使われるものです）．

シングルクリックかダブルクリックかのチェックは1回目のボタン開放からある一定時間を経過して押下がなければ，シングルクリックであると判断します．

サンプルプログラムではこの一定待ち時間を変数intervalによって調整します．これは使用している機種の速さによって多少変化しますからみなさんがちょうどいいと思われる値に変更してください（ダブルクリックが失敗しやすいときはこの値を大きめにする）．

このプログラムは原理的なものを確認するためのものなので，みなさんがいろいろ改良してみてください（待ち時間の機種間の違いなど）．

```
#include<stdio.h>
#include<dos.h>
#include<stdlib.h>

main()
{
        int  i,interval=2500;          /*この値を調整してください*/
        union  REGS  regs;

/*マウスドライバのチェックおよび初期化*/
        regs.x.ax  =  0;                       /*機能コード00H*/
        int86(0x33,&regs,&regs);
        if  (regs.x.ax  ==  0){
                puts("マウスが使えなーい");
                exit(1);
        }

/*マウスカーソルの表示*/
        regs.x.ax  =  1;                       /*機能コード01H*/
        int86(0x33,&regs,&regs);

/*チェック部　右ボタンを押すと止まります*/
        do{

                do{                                                          /*1回目の押下待ち* /
                        regs.x.ax  =  3;
                        int86(0x33,&regs,&regs);
                } while  (regs.x.ax  ==  0  &&  regs.x.bx  ==  0);
                while(regs.x.ax  !=  0  &&  regs.x.bx  ==  0){        /*1回目の開放待ち*/
                        regs.x.ax  =  3;
                        int86(0x33,&regs,&regs);
                }
```

```
        for(i=0;i<interval && regs.x.bx == 0;i++){     /*2回目の押下または*/
            regs.x.ax = 3;                              /*時間切れ待ち     */
            int86(0x33,&regs,&regs);
            if(regs.x.ax != 0){
                break;
            }
        }
        while (regs.x.ax != 0 && regs.x.bx == 0){       /*2回目の開放待ち*/
            regs.x.ax = 3;
            int86(0x33,&regs,&regs);
        }

        if(regs.x.bx == 0){
            if(i<interval){
                puts("ダブルクリックです");
            }else{
                puts("シングルクリックです");
            }
        }

    } while(regs.x.bx == 0);

/*マウスカーソル消去*/
    regs.x.ax = 2;                      /*機能コード02H*/
    int86(0x33,&regs,&regs);

    return 0;
}
```

●ボタン状態の情報の取得

これはサンプルプログラムではなく，ボタン状態の情報を取得する各種機能を使用したサンプルルーチンです．一定時間ごとにマウスの左ボタンの状態，ボタンを押した回数，最後にボタンが押された時の座標を表示します．何かキーを入力するとプログラムが終了します．

```
    union REGS regs;
    int  i,key;

    do{
        regs.x.ax = 0x05;                           /*機能コード05H*/
        int86(0x33,&regs,&regs);
       printf("x = %3d , y = %3d ,現在の左ボタン状態 = %2d,押下回数 = %2d \n"
                          ,regs.x.cx,regs.x.dx,regs.x.ax,regs.x.bx);

        for(i=0;i<3000;i++){                              /*一定時間待ち*/
            if ((key=kbhit())!=0){ break; }    /*キーボード入力でループをぬける*/
        }

    } while (!key);
```

■マウスBIOS（マイクロソフト仕様［PC-9801シリーズ用］）一覧（INT 33H）

コード	機　　能	ＮＥＣ仕様相違
００Ｈ	マウスＢＩＯＳ初期化	初期化項目
０１Ｈ	マウスカーソル表示	
０２Ｈ	マウスカーソル消去	回数スタックなし
０３Ｈ	マウスカーソル位置，ボタン状態取得	ボタンの情報形式
０４Ｈ	マウスカーソル位置設定	
０５Ｈ	ボタン押下情報取得	左右独立コード
０６Ｈ	ボタン開放情報取得	左右独立コード
０７Ｈ	水平方向のマウスカーソル移動範囲設定	機能コード１０Ｈ
０８Ｈ	垂直方向のマウスカーソル移動範囲設定	機能コード１１Ｈ
０９Ｈ	マウスカーソル形状設定	形状データ形式
０ＢＨ	マウスカーソルの移動距離取得	
０ＣＨ	ユーザー定義サブルーチンのコール条件設定	
０ＦＨ	ミッキー／ドット比設定	
ＦＤＨ	グラフィックＶＲＡＭ設定と実装状況取得	機能コード１３Ｈ
ＦＥＨ	マウス割り込みの許可	機能コード１４Ｈ
ＦＦＨ	カーソル表示画面設定	機能コード１２Ｈ

1 マウスBIOS初期化 MS

割り込み INT　33H

入　　力 AH←00H

出　　力 AX→マウスの環境状態　0：使用不可能
　　　　　　　　　　　　　　　　 −1：使用可能

BX→ボタンの数

解　　説 マウスが使用可能であるかどうかをチェックし，使用可能であればマウス環境を初期化します．このコマンドにより以下のすべての項目が初期化されます．

ドライバ組み込み時の初期状態

カーソル表示*	表示しない
カーソル位置*	(319, 199) 400ライン時 (319, 99) 200ライン時
カーソル表示画面*	0, 1, 2プレーン
カーソル移動範囲*	画面全体 (0, 0) ~ (639, 199) 400ライン時 (0, 0) ~ (639, 99) 200ライン時
カーソル形状*	左上向きの矢印 (16×16ドット)
カーソル中心点*	(0, 0)
ミッキー/ドット比*	水平方向8, 垂直方向8

サンプル マウスBIOSを初期化します.

```
union REGS regs;

regs.x.ax = 0x00;                          /*機能コード00H*/
int86(0x33,&regs,&regs);
if (regs.x.ax == 0){
      puts("マウスが使えなーい");
      exit(1);
} else {
      puts("マウスドライバを初期化したぞ");
}
```

2 マウスカーソル表示 MS

割り込み INT 33H

入　力 AX←01H

出　力 なし

解　説 マウスカーソルを画面上に表示します. このコマンドを実行すると, カーソルの消去またはマウスBIOS初期化が実行されるまで表示し続けます.

サンプル マウスカーソルを表示します.

```
union REGS regs;

regs.x.ax = 0x01;                          /*機能コード01H*/
int86(0x33,&regs,&regs);
```

3 マウスカーソル消去 MS

割り込み INT　33H

入　力 AX←02H

出　力 なし

解　説 マウスカーソルを画面上より消去します．このコマンドを実行すると，マウスカーソルの表示が再び実行されるまでカーソルは表示されません．ただし，表示されていない間もマウス本体の移動にあわせてのカーソルは画面上を移動しています．したがって再びカーソルを表示させたときは非表示の間に移動したカーソル位置に表示されます．

【注】このコマンドを複数回実行したときは，同じ回数だけ機能コード01H（マウスカーソル表示）を実行しないとカーソルは表示されません．

サンプル マウスカーソルを消去します．

```
union  REGS  regs;

regs.x.ax  =  0x02;                         /*機能コード02H*/
int86(0x33,&regs,&regs);
```

4 マウスカーソル位置，ボタン状態取得 MS

割り込み INT　33H

入　力 AX←03H

出　力 BX→ボタンの状態

bit0：左ボタンの状態
bit1：右ボタンの状態
各ビット1のときに押されている

CX→カーソル位置の水平座標　0～639

DX→カーソル位置の垂直座標　0～399：400ライン時
0～199：200ライン時

解　説 マウスカーソルの位置とボタンの状態を取得します．このコマンドによりカーソルの水平，垂直の両座標，並びに左右ボタンの状態をすべて一度に得ることができます．

サンプル マウスカーソル位置，ボタン状態の取得し，画面に表示します．マウスの左ボタンを押すと終了します．

```
union  REGS  regs;

do{
      regs.x.ax  =  3;              /*機能コード03H*/
      int86(0x33,&regs,&regs);
      printf("x  =  %3d  ,y  =  %3d  ,ボタン  =  %2d  \r"
            ,regs.x.cx,regs.x.dx,regs.x.bx);
} while  ((regs.x.bx  &  0x0001)  ==  0  );  /*ビット0以外のマスク*/
```

5 マウスカーソル位置設定 MS

割り込み INT　33H

入　力 AX←04H

CX←新たに設定するカーソル位置の水平座標

DX←新たに設定するカーソル位置の垂直座標

出　力 なし

解　説 マウスカーソルの位置を新たに任意の位置に移動します．設定できる座標は画面のモードと機能コード11H（カーソル移動範囲の設定）により設定された範囲内で決まります．また，この範囲を超えた場合，移動範囲内の端にカーソルは移動します．

サンプル カーソルをディスプレイの中心に設定します．

```
union  REGS  regs;

regs.x.ax  =  0x04;                 /*機能コード04H*/
regs.x.cx  =  320;                  /*カーソルX座標指定*/
regs.x.dx  =  200;                  /*カーソルY座標指定*/
int86(0x33,&regs,&regs);
```

6 ボタン押下情報取得 MS

割り込み INT　33H

入　力 AX←05H

BX←情報を得たいボタンの指定　0：左ボタン

1：右ボタン

サンプル AX→ボタンの状態

bit0：左ボタンの状態

bit1：右ボタンの状態

各ビット1のときに押されている

BX→前回呼び出し後からの押された回数

CX→最後に押されたときのカーソル位置の水平座標

DX→最後に押されたときのカーソル位置の垂直座標

解説 指定されたボタンが押されたときに関する各種情報を取得します．ボタンの状態は機能コード03H（マウスカーソル位置，ボタン状態取得）でも得られますが，このコマンドではさらに，最後にボタンが押されたときのカーソルの座標およびこのコマンドが前回呼び出された後から左ボタンが押された回数を取得することができます．したがって，このコマンドを用いれば常時マウスの状態を追いかけることなくボタンを押したときのカーソルの座標を読み取ることができます．

サンプル 一定時間ごとにマウスの左ボタンの状態，ボタンを押した回数，最後にボタンが押された時の座標を表示します．何かキーを入力するとプログラムが終了します．

```
union  REGS  regs;
int  i,key;

do{
      regs.x.ax  =  0x05;                          /*機能コード05H*/
      regs.x.bx  =  0                               /*左ボタンを指定*/
      int86(0x33,&regs,&regs);
      printf("x=%3d,  y=%3d,現在のボタン状態=%2d,押下回数  =%2d  \n"
                ,regs.x.cx,regs.x.dx,regs.x.ax,regs.x.bx);

      for(i=0;i<3000;i++){                              /*一定時間待ち*/
        if((key=kbhit())!=0){break;}/*キーボード入力でループをぬける*/
      }

} while  (!key);
```

7 ボタン開放情報取得 MS

割り込み INT　33H

入　力 AX←06H

BX←情報を得たいボタンの指定　0：左ボタン

1：右ボタン

出　力 AX→ボタンの状態

bit0：左ボタンの状態

bit1：右ボタンの状態

各ビット1のときに押されている

BX→前回呼び出し後からの離された回数

CX→最後に離されたときのカーソル位置の水平座標

DX→最後に離されたときのカーソル位置の垂直座標

解　説　指定されたボタンが離されたときに関する各種情報を取得します．ボタンの状態は機能コード03H（マウスカーソル位置，ボタン状態取得）でも得られますが，このコマンドではさらに，最後にボタンが離されたときのカーソルの座標およびこのコマンドが前回呼び出された後から左ボタンが離された回数を取得することができます．このコマンドは機能コード05H（ボタン押下状態取得）において，〈ボタンを押したときの状態〉が〈離されたときの状態〉に変わったものと考えられます．

サンプル　機能コード05Hのサンプルのを参照してください．

8 水平方向のマウスカーソル移動範囲設定 MS

割り込み　INT　33H

入　力　AX←07H

CX←カーソルの水平方向の移動範囲の最小値　0〜639

DX←カーソルの水平方向の移動範囲の最大値　0〜639

出　力　なし

解　説　マウスカーソルの水平方向の画面上移動範囲を設定します．ここで設定された移動範囲がカーソルの中心点が移動できる範囲となります（p.360,図2-40）．なお，CXレジスタの値がDXレジスタより大きかった場合CXが最大値，DXが最小値となります．

サンプル　カーソルの水平方向の移動範囲を100〜540に設定します．

```
union  REGS  regs;

regs.x.ax  =  0x07;                /*機能コード07H*/
regs.x.cx  =  100;
regs.x.dx  =  540;
int86(0x33,&regs,&regs);        /*X座標範囲の設定*/
```

9 垂直方向のマウスカーソル移動範囲設定 MS

割り込み　INT　33H

入　力　AX←08H

CX←カーソルの垂直方向の移動範囲の最小値　0〜399：400ライン時

0〜199：200ライン時

DX←カーソルの垂直方向の移動範囲の最大値　0～399：400ライン時
0～199：200ライン時

出力 なし

解説 マウスカーソルの垂直方向の画面上移動範囲を設定します．ここで設定された移動範囲がカーソルの中心点が移動できる範囲となります（図2-40）．なお，CXレジスタの値がDXレジスタより大きかった場合CXが最大値，DXが最小値となります．

サンプル カーソルの垂直方向の移動範囲を50～350に設定します．

```
union  REGS  regs;

regs.x.ax  =  0x08;                    /*機能コード08H*/
regs.x.cx  =  50;
regs.x.dx  =  350;
int86(0x33,&regs,&regs);          /*Y座標範囲の設定*/
```

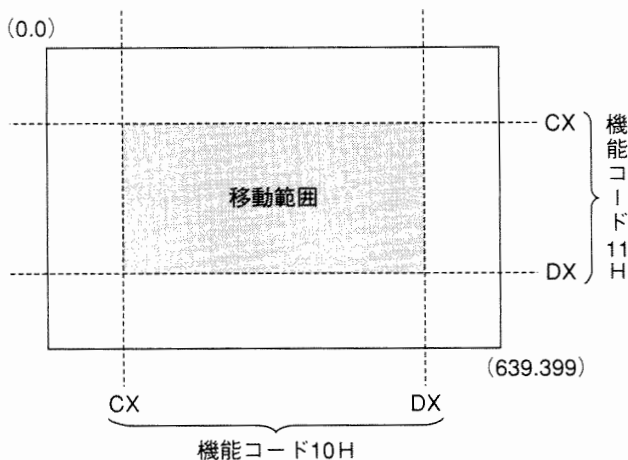



図2-40 マウスの移動範囲

10 マウスカーソル形状設定 MS

割り込み INT　33H

入力 AX←09H

BX←カーソルの中心点の水平座標　0～15

CX←カーソルの中心点の垂直座標　0～15

ES：DX←カーソル形状データの先頭アドレス
データ形式は16×16×2ビット

出　力　なし

解　説

マウスカーソルの形状および中心点を設定します．マウスカーソルは16×16ドットのデータで形成されており，中心点とはそのカーソルデータ内のどのドットがマウスの指し示す座標となるかを決定するものです．中心点はカーソル形状データの左上を（0，0），右下を（15，15）としたとき，この範囲内で自由に設定することができます．初期状態ではカーソル形状は左上向きの矢印となっていますから矢印の先端（0，0）に中心点が設定されています（図2-41）．

マウスカーソルの形状データのフォーマットはカーソル形状データの座標で（0，0），（1，0），……，（15，0），（0，1），（2，1），……，（15，15）の順に1ビットずつ並べたものを2つ（AND用とXOR用の順）連続させたものです．実際の表示はまず，AND用のカーソル形状データと画面上（VRAM上）に表示されているデータとのANDが行われます．次にそのANDをとったデータとXOR用のカーソル形状データとのXORが行われ，これが画面上に表示されます．この表示は各プレーンごとに対して行われます（ただし，カーソル形状データは各プレーン共通）．

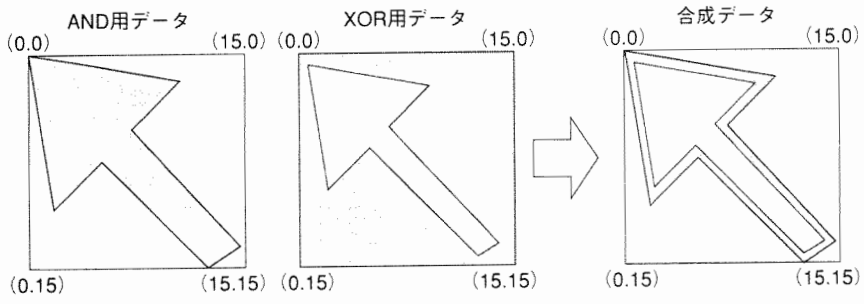



図2-41 マウスのデータ形式（マイクロソフト）

サンプル　カーソルの形状を縁取り付きの真上向き矢印にします．

```
int  keijou[16*2]={
/*AND  DATA*/
        0xfeff,                 /*1111111011111111*/
        0xfc7f,                 /*1111110001111111*/
        0xf83f,                 /*1111100000111111*/
        0xf01f,                 /*1111000000011111*/
        0xe00f,                 /*1110000000001111*/
        0xc007,                 /*1100000000000111*/
        0x8003,                 /*1000000000000011*/
        0x0001,                 /*0000000000000001*/
        0xf83f,                 /*1111100000111111*/
        0xf83f,                 /*1111100000111111*/
        0xf83f,                 /*1111100000111111*/
        0xf83f,                 /*1111100000111111*/
        0xf83f,                 /*1111100000111111*/
        0xf83f,                 /*1111100000111111*/
        0xf83f,                 /*1111100000111111*/
```

```
        0xf83f,                         /*1111100000111111*/
/*XOR DATA*/
        0x0000,                         /*0000000000000000*/
        0x0100,                         /*0000000100000000*/
        0x0380,                         /*0000001110000000*/
        0x07c0,                         /*0000011111000000*/
        0x0fe0,                         /*0000111111100000*/
        0x1ff0,                         /*0001111111110000*/
        0x3ff8,                         /*0011111111111000*/
        0x0380,                         /*0000001110000000*/
        0x0380,                         /*0000001110000000*/
        0x0380,                         /*0000001110000000*/
        0x0380,                         /*0000001110000000*/
        0x0380,                         /*0000001110000000*/
        0x0380,                         /*0000001110000000*/
        0x0380,                         /*0000001110000000*/
        0x0380,                         /*0000001110000000*/
        0x0000,                         /*0000000000000000*/
};

union REGS regs;

regs.x.ax = 0x09;                       /*機能コード09H*/
regs.x.bx = 7;                          /*カーソル中心点の水平座標*/
regs.x.cx = 0;                          /*カーソル中心点の垂直座標*/
regs.x.dx = (int)keijou;                /*カーソル形状データのアドレス設定*/
int86(0x33,&regs,&regs);
```


11 マウスカーソルの移動距離取得 MS

割り込み INT 33H

入　力 AX←0BH

出　力 CX→マウスの水平方向移動距離　-32768～32767[ミッキー]
DX→マウスの垂直方向移動距離　-32768～32767[ミッキー]

解　説 マウスの移動距離を取得します．このコマンドが前回実行された時から今回実行される時までのマウス位置の相対的な移動距離がミッキー単位で出力されます．ここで水平方向は右向き，垂直は下向きが正の向きとなります．

サンプル 前回左ボタンが押された時から今回左ボタンを押した時までのマウスの移動量を画面に表示します．右ボタンで終了します．

```
union REGS regs;

while(1){
        do{                                     /*ボタンの入力待ち*/
                regs.x.ax = 0x03;
```

```
        int86(0x33,&regs,&regs);
    } while  (  !  regs.x.bx  );
    if(regs.x.bx  &&  0x0002){  break;  }

    regs.x.ax  =  0x0b;                     /*機能コード0BH*/
    int86(0x33,&regs,&regs);
    printf("x  =  %4d  ,  y  =  %4d  \r",regs.x.cx,regs.x.dx);

    do{                                     /*ボタンの開放待ち*/
        regs.x.ax  =  0x03;
        int86(0x33,&regs,&regs);
        } while  (regs.x.bx);
}
```

12 ユーザー定義サブルーチンのコール条件設定 MS

割り込み INT 33H

入力 AX←0CH

CX←コール条件

ビット0：マウスカーソルの位置変化
ビット1：左ボタンの押下
ビット2：左ボタンの開放
ビット3：右ボタンの押下
ビット4：右ボタンの開放

各ビット1のときにコールする
複数指定可能

ES：DX←ユーザー定義サブルーチンのアドレス

出力 なし

解説 ユーザーが作成したサブルーチンをマウスドライバがコールする条件とそのアドレスを設定します．このコールは次の手順で行われます．まずマウス割り込みによって制御がマウスドライバに移ります．次に，マウスドライバは指定されたコール条件のうち1つでも条件を満たせばサブルーチンをコールします．ここで，マウスドライバからのコールはユーザー定義サブルーチンがFAR型プロシージャとして行われます．したがってユーザーはサブルーチンをFAR型のプロシージャとして作成してください．

また，マウスドライバがサブルーチンをコールする際には次のような情報をレジスタに格納してコールします．よって，ユーザー定義サブルーチン内ではこれらのレジスタを参照することによりマウスの情報を利用できます．

AX→コールの原因となった現象
1：カーソルの位置変化
2：左ボタンの押下
4：左ボタンの開放
8：右ボタンの押下
16：右ボタンの開放
BX→ボタンの状態
ビット0：左ボタン
ビット1：右ボタン
各ビット1の時，押されている
CX→カーソル位置の水平座標
DX→カーソル位置の垂直座標

13 ミッキー／ドット比設定 MS

割り込み INT 33H

入　力 AX←0FH
CX←水平方向のミッキー／ドット比
DX←垂直方向のミッキー／ドット比

出　力 なし

解　説 マウス本体の移動距離（ミッキー単位）とそれに対応する画面上のマウスカーソルの移動距離（ドット単位）の比を設定します．この設定はマウスカーソルを8ドット移動させるのに要するマウス本体の移動距離（ミッキー／8ドット，1ミッキーは約0.25mm）を単位として設定します（水平，垂直方向はそれぞれ個別に設定可能）．

したがってこの値を大きくすればマウスの感度は低くなり，値を小さくすればマウスの感度が高くなります．また，負の値を入れることにより，マウスの移動方向を反転させることもできます．

サンプル 水平方向のミッキー／ドット比を4，垂直方向のミッキー／ドット比を16に設定します．

```
union REGS regs;

regs.x.ax = 0x0f; /*機能コード0FH（ミッキー／ドット比設定）*/
regs.x.cx =   4;  /*水平方向のミッキー／ドット比          */
regs.x.dx =  16;  /*水平方向のミッキー／ドット比          */
int86(0x33,&regs,&regs);
```

14 グラフィックVRAM設定と実装状況取得 MS

割り込み　INT　33H

入　力　AX←FDH
BX←グラフィックVRAMの使用画面設定
0：0～2プレーン使用
1：0～3プレーン使用

出　力　BX→グラフィックVRAM実装状態
0：プレーン3を実装してない
−1：プレーン3を実装している

解　説　グラフィックVRAMのプレーン3の使用設定と実装状況の取得をします．プレーン3は16色に対応していない機種（16色ボードが実装されているものは除く）には存在しないため使用できません．また，16色対応機種でもプレーン3はこのコマンドで使用を宣言しないと利用できません．

【注】この機能コードは古いバージョンのMOUSE.COMでは使用できません．

サンプル　プレーン3の実装チェックを行い，使用プレーンを0～3に設定をします．

```
union  REGS  regs;

regs.x.ax  =  0xfd;                    /*機能コードFDH*/
regs.x.bx  =  1;                        /*プレーン0～3を設定*/
int86(0x33,&regs,&regs);
if  (regs.x.bx  ==  0){
        puts("プレーン3がないぞ");
}
```

15 マウス割り込みの許可 MS

割り込み　INT　33H

入　力　AX←FEH
BX←マウス割り込み（8255Aの割り込み）制御
0：割り込み停止
−1：割り込み開始

出　力　なし

解　説　マウス割り込みを停止，開始させます．

16 カーソル表示画面設定

割り込み INT 33H

入　力 AX←FFH

BX←マウスカーソル表示画面

bit0：プレーン0
bit1：プレーン1
bit2：プレーン2
bit3：プレーン3
各ビット1の時，表示する

出　力 なし

解　説 カーソルの表示画面を設定します．指定したすべてのプレーンに対しカーソルを表示します．

プレーン3は16色に対応していない機種（16色ボードが実装されているものは除く）には存在しないため，設定しても表示されません．また，16色に対応している機種でも機能コードFDH（グラフィックVRAM設定と実装状況取得）で，プレーン3の使用を宣言しないと利用できませんので注意してください．

サンプル カーソル表示画面を0～3のすべてのプレーンに設定します．

```
union  REGS  regs;

regs.x.ax  =  0xff;              /*機能コードFFH*/
regs.x.bx  =  0x000f;            /*0x0f=00001111B*/
int86(0x33,&regs,&regs);
```

■サンプルプログラム

●カーソルの表示

まずは，機能コード0～03Hまでを使ったプログラムです．MOUSE.COMの組み込みを忘れないでください．

実行させるとマウスカーソルが表示されて画面内を自由に動かせます．左ボタンを押すことにより停止します．最後にカーソルの消去を行います．

```
#include<stdio.h>
#include<dos.h>
#include<stdlib.h>

main()
{
```

```
	union  REGS  regs;

/*マウスドライバのチェックおよび初期化*/
	regs.x.ax  =  0;			/*機能コード00H*/
	int86(0x33,&regs,&regs);
	if  (regs.x.ax  ==  0){
		puts("マウスが使えなーい");
		exit(1);
	} else  {
		puts("マウスドライバを初期化したぞ");
	}

/*マウスカーソルの表示*/
	regs.x.ax  =  1;			/*機能コード01H*/
	int86(0x33,&regs,&regs);

/*マウスカーソル位置，ボタン状態の取得*/
	do{
		regs.x.ax  =  3;		/*機能コード03H*/
		int86(0x33,&regs,&regs);
		printf("x  =  %3d  ,y  =  %3d  ,ボタン  =  %2d  \r"
			,regs.x.cx,regs.x.dx,regs.x.bx);
	} while  ((regs.x.bx  &  0x0001)  ==  0  );  /*ビット0以外のマスク*/
	/*左ボタンを押すと止まります*/

/*マウスカーソル消去*/
	regs.x.ax  =  2;			/*機能コード02H*/
	int86(0x33,&regs,&regs);

	return(0);
}
```

●カーソル形状変更

カーソル形状を変更するプログラムを示します．カーソルを縁取り付きの真上向き矢印にします．真上向きなので左利きの人も違和感なく使えると思います．また，このプログラムではカーソルを黒い縁取りにするため，プレーン3も使用するようにしました．みなさんも自分で気に入った形や，中心点に書き換えてみてください．

```
#include<stdio.h>
#include<dos.h>
#include<stdlib.h>

int  keijou[16*2]={
/*AND  DATA*/
	0xfeff,			/*1111111011111111*/
	0xfc7f,			/*1111110001111111*/
	0xf83f,			/*1111100000111111*/
	0xf01f,			/*1111000000011111*/
	0xe00f,			/*1110000000001111*/
	0xc007,			/*1100000000000111*/
	0x8003,			/*1000000000000011*/
	0x0001,			/*0000000000000001*/
	0xf83f,			/*1111100000111111*/
```

```
        0xf83f,                         /*1111100000111111*/
        0xf83f,                         /*1111100000111111*/
        0xf83f,                         /*1111100000111111*/
        0xf83f,                         /*1111100000111111*/
        0xf83f,                         /*1111100000111111*/
        0xf83f,                         /*1111100000111111*/
        0xf83f,                         /*1111100000111111*/
/*XOR  DATA*/
        0x0000,                         /*0000000000000000*/
        0x0100,                         /*0000000100000000*/
        0x0380,                         /*0000001110000000*/
        0x07c0,                         /*0000011111000000*/
        0x0fe0,                         /*0000111111100000*/
        0x1ff0,                         /*0001111111110000*/
        0x3ff8,                         /*0011111111111000*/
        0x0380,                         /*0000001110000000*/
        0x0380,                         /*0000001110000000*/
        0x0380,                         /*0000001110000000*/
        0x0380,                         /*0000001110000000*/
        0x0380,                         /*0000001110000000*/
        0x0380,                         /*0000001110000000*/
        0x0380,                         /*0000001110000000*/
        0x0380,                         /*0000001110000000*/
        0x0000,                         /*0000000000000000*/
};

main()
{
        union  REGS  regs;

/*マウスドライバのチェックおよび初期化*/
        regs.x.ax  =  0x00;                     /*機能コード00H*/
        int86(0x33,&regs,&regs);
        if  (regs.x.ax  ==  0){
                puts("マウスが使えなーい");
                exit(1);
        }

/*プレーン3設定*/
        regs.x.ax  =  0xfd;         /*機能コードFDH*/
        regs.x.bx  =  1;
        int86(0x33,&regs,&regs);
        if  (regs.x.bx  !=  0){
                regs.x.ax  =  0xff;         /*機能コードFFH*/
                regs.x.bx  =  0x000f;
                int86(0x33,&regs,&regs);
        }

/*マウスカーソル形状設定*/
        regs.x.ax  =  0x09;                     /*機能コード09H*/
        regs.x.bx  =  7;                          /*カーソル中心点の水平座標*/
        regs.x.cx  =  0;                          /*カーソル中心点の垂直座標*/
        regs.x.dx  =  (int)keijou;      /*カーソル形状データのアドレス設定*/
        int86(0x33,&regs,&regs);
```

```
/*マウスカーソルの表示*/
      regs.x.ax  =  0x01;                  /*機能コード01H*/
      int86(0x33,&regs,&regs);

/*左ボタンが押されるまでループ*/
      do{
             regs.x.ax  =  0x03;
             int86(0x33,&regs,&regs);
      } while  ((regs.x.bx  &  0001)  ==  0);

/*マウスカーソル消去*/
      regs.x.ax  =  0x02;                  /*機能コード02H*/
      int86(0x33,&regs,&regs);

      return  (0);
}
```

●そのほかのサンプルプログラム

NEC仕様マウスドライバのサンプルプログラムと大きな差異はないので，そちらの方を参考にしてください．

■2-11-5 ハイレゾモードマウスBIOS

■マウスBIOS（ハイレゾモードROM BIOS）一覧（INT 33H）

コード	機　　　能	MOUSE.SYSとの相違
００H	マウスＢＩＯＳ初期化	
０１H	マウスカーソル表示	
０２H	マウスカーソル消去	
０３H	マウスカーソル位置，ボタン状態取得	ボタンの情報形式
０４H	マウスカーソル位置設定	
０５H	ボタン押下情報取得	左右独立コード
０６H	ボタン開放情報取得	左右独立コード
０７H	水平方向のマウスカーソル移動範囲設定	機能コード１０H
０８H	垂直方向のマウスカーソル移動範囲設定	機能コード１１H
０９H	マウスカーソル形状設定	形状データ形式
０BH	マウスカーソルの移動距離取得	
０CH	ユーザー定義サブルーチンのコール条件設定	
０FH	ミッキー／ドット比設定	
１０H	マウスカーソル表示画面，表示パターン設定	機能コード１２H
１１H	マウス割り込みアドレスの設定	該当機能なし
１２H	マウス割り込みの許可	機能コード１４H

1 マウスBIOS初期化

割り込み INT 33H

入力 AH←00H

出力 AX→マウスの環境状態　0：使用不可能
　　　　　　　　　　　　　−1：使用可能

BX→ボタンの数

解説

マウスが使用可能であるかどうかをチェックし，使用可能であればマウス環境を初期化します．

ドライバ組み込み時の初期状態を以下に示しますが，このコマンドにより*のついた項目が再初期化されます（*がついていない項目はこのコマンドでは初期化されない）．

ドライバ組み込み時の初期状態

カーソル表示*	表示しない
カーソル位置	(512，384)
カーソル表示画面	0〜3プレーン
カーソル移動範囲	画面全体 (0，0)〜(1119，935)
カーソル形状*	左上向きの矢印
カーソル中心点*	(0，0)
ミッキー／ドット比*	水平方向8，垂直方向8
ユーザー定義サブルーチンのコール条件*	0
マウス割り込み周期	8ms

2 マウスカーソル表示

割り込み INT 33H

入力 AX←01H

出力 なし

解説

マウスカーソルを画面上に表示します．このコマンドを実行すると，カーソルの消去またはマウスBIOS初期化が実行されるまで表示し続けます．

3 マウスカーソル消去 ハイレゾ

割り込み INT 33H

入 力 AX←02H

出 力 なし

解 説 マウスカーソルを画面上より消去します．このコマンドを実行すると，マウスカーソルの表示が再び実行されるまでカーソルは表示されません．ただし，表示されていない間もマウス本体の移動にあわせてのカーソルは画面上を移動しています．したがって再びカーソルを表示させたときは非表示の間に移動したカーソル位置に表示されます．

4 マウスカーソル位置，ボタン状態取得 ハイレゾ

割り込み INT 33H

入 力 AX←03H

BX→ボタンの状態

bit0：左ボタンの状態

bit1：右ボタンの状態

各ビット1のときに押されている

出 力 CX→カーソル位置の水平座標 0〜1119

DX→カーソル位置の垂直座標 0〜935

解 説 マウスカーソルの位置とボタンの状態を取得します．このコマンドによりカーソルの水平，垂直の両座標，並びに左右ボタンの状態をすべて一度に得ることができます．

5 マウスカーソル位置設定 ハイレゾ

割り込み INT 33H

入 力 AX←04H

CX←新たに設定するカーソル位置の水平座標

DX←新たに設定するカーソル位置の垂直座標

出 力 なし

解 説 マウスカーソルの位置を新たに任意の位置に移動します．設定できる座標は画面のモ

ードと機能コード11H（カーソル移動範囲の設定）により設定された範囲内で決まります．また，この範囲を超えた場合，移動範囲内の端にカーソルは移動します．

6　ボタン押下情報取得　ハイレゾ

割り込み　INT　33H

入　力　AX←05H
BX←情報を得たいボタンの指定　0：左ボタン
1：右ボタン

出　力　AX→ボタンの状態
bit0：左ボタンの状態
bit1：右ボタンの状態
各ビット1のときに押されている
BX→前回呼び出し後からの押された回数
CX→最後に押されたときのカーソル位置の水平座標
DX→最後に押されたときのカーソル位置の垂直座標

解　説　指定されたボタンが押されたときに関する各種情報を取得します．ボタンの状態は機能コード03H（マウスカーソル位置，ボタン状態取得）でも得られますが，このコマンドではさらに，最後にボタンが押されたときのカーソルの座標およびこのコマンドが前回呼び出された後から左ボタンが押された回数を取得することができます．したがって，このコマンドを用いれば常時マウスの状態を追いかけることなくボタンを押したときのカーソルの座標を読み取ることができます．

7　ボタン開放情報取得　ハイレゾ

割り込み　INT　33H

入　力　AX←06H
BX←情報を得たいボタンの指定　0：左ボタン
1：右ボタン

出　力　AX→ボタンの状態
bit0：左ボタンの状態
bit1：右ボタンの状態
各ビット1のときに押されている

BX→前回呼び出し後からの離された回数

CX→最後に離されたときのカーソル位置の水平座標

DX→最後に離されたときのカーソル位置の垂直座標

解　説　指定されたボタンが離されたときに関する各種情報を取得します．ボタンの状態は機能コード03H（マウスカーソル位置，ボタン状態取得）でも得られますが，このコマンドではさらに，最後にボタンが離されたときのカーソルの座標およびこのコマンドが前回呼び出された後から左ボタンが離された回数を取得することができます．このコマンドは機能コード05H（ボタン押下状態取得）において，〈ボタンを押したときの状態〉が〈離されたときの状態〉に変わったものと考えられます．

8 水平方向のマウスカーソル移動範囲設定 ハイレゾ

割り込み　INT　33H

入　力　AX←07H

CX←カーソルの水平方向の移動範囲の最小値　0〜1119

DX←カーソルの水平方向の移動範囲の最大値　0〜1119

出　力　なし

解　説　マウスカーソルの水平方向の画面上移動範囲を設定します．ここで設定された移動範囲がカーソルの中心点が移動できる範囲となります（図2-42）．なお，CXレジスタの値がDXレジスタより大きかった場合CXが最大値，DXが最小値となります．この設定により現在のカーソル位置が移動範囲外となった場合，カーソルは自動的に移動範囲内の端に移動されます．

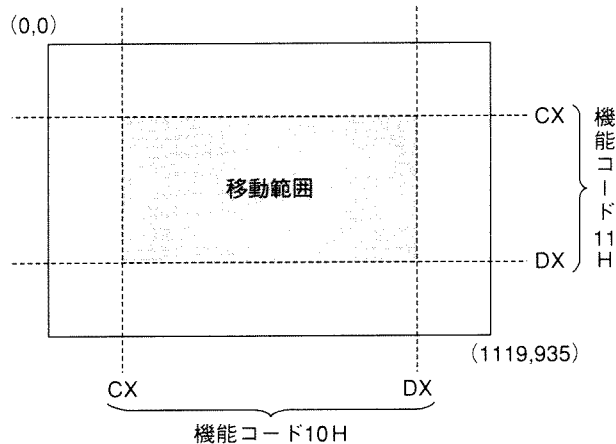



図2-42　マウスの移動範囲

9 垂直方向のマウスカーソル移動範囲設定 ハイレゾ

割り込み INT 33H

入 力 AX←08H

CX←カーソルの垂直方向の移動範囲の最小値 0～935

DX←カーソルの垂直方向の移動範囲の最大値 0～935

出 力 なし

解 説 マウスカーソルの垂直方向の画面上移動範囲を設定します．ここで設定された移動範囲がカーソルの中心点が移動できる範囲となります．なお，CXレジスタの値がDXレジスタより大きかった場合CXが最大値，DXが最小値となります．この設定により現在のカーソル位置が移動範囲外となった場合，カーソルは自動的に移動範囲内の端に移動されます．

10 マウスカーソル形状設定 ハイレゾ

割り込み INT 33H

入 力 AX←09H

BX←カーソルの中心点の水平座標 −16～32

CX←カーソルの中心点の垂直座標 −16～48

ES：DX←カーソル形状データの先頭アドレス

SI←カーソル形状データの大きさ設定

上位：横方向ビット数（8ビット単位で設定）

下位：縦方向ビット数（8ビット単位で設定）

（8×8ビット～32×32ビット）

出 力 なし

解 説 マウスカーソルの形状および中心点を設定します．マウスカーソルは8×8～32×32ドットのデータで形成されており，中心点とはそのカーソルデータ内のどのドットがマウスの指し示す座標となるかを決定するものです．中心点はカーソル形状データの左上を（0，0）としたときの座標で設定することができます．初期状態ではカーソル形状は左上向きの矢印となっていますから矢印の先端（0，0）に中心点が設定されています．

マウスカーソル形状データのフォーマットは16×16ドットの場合カーソル形状データの座標で（0，0），（1，0），……，（15，0），（0，1），（2，1），……，（15，15）の順に1ビットずつ並べたものを単位とします．この形状データの大きさはSIレジ

スタの上位にカーソルブロックの横ドット数，下位ビットに縦ドット数を入れることにより指定します（縦横ともに8ビット単位で設定でき，最小は8×8，最大は32×32ビット）．

実際のカーソル表示は上記のデータ形状と画面上に表示されているデータとのAND，XORをとった形となります（ANDとXORを両方とる場合はANDをとった後にXORをとる）．

さらにデータ形状の設定は各プレーンごとに行うことができます．設定する際のカーソル形状データの順番（上に示したデータ形式を1ブロックとして）をいくつかの例を上げて説明します．

・1画面にXORのみをとる場合

指定プレーンのXOR用データ

・1画面のみにAND，XORをとる場合

指定プレーンのAND用データ→指定プレーンのXOR用データ

・4画面異なったカーソル形状データでXORのみをとる場合

0プレーンのXOR用データ→1プレーンのXOR用データ
→2プレーンのXOR用データ→3プレーンのXOR用データ

・4画面に異なったカーソル形状データでAND，XORをとる場合

0プレーンのAND用データ→0プレーンのXOR用データ
→1プレーンのAND用データ→1プレーンのXOR用データ
→2プレーンのAND用データ→2プレーンのXOR用データ
→3プレーンのAND用データ→3プレーンのXOR用データ

これらのデータ形式パターンの設定は機能コード10H（カーソル表示画面，表示パターンの設定）により行います．したがってこれらのパターンを変更する場合，このコマンドの実行に先立って機能コード10Hのコマンドを実行しなければなりません．

11 マウスカーソルの移動距離取得

割り込み INT 33H

入力 AX←0BH

出力 CX→マウスの水平方向移動距離　−1119〜1119[ミッキー]
DX→マウスの垂直方向移動距離　−935〜935[ミッキー]

解説 マウスの移動距離を取得します．このコマンドが前回実行された時から今回実行される時までのマウス位置の相対的な移動距離がミッキー単位で出力されます．ここで水平方向は右向き，垂直は下向きが正の向きとなります．

12 ユーザー定義サブルーチンのコール条件設定 ハイレゾ

割り込み　INT　33H

入　力　AX←0CH

CX←コール条件

ビット0：マウスカーソルの位置変化
ビット1：左ボタンの押下
ビット2：左ボタンの開放
ビット3：右ボタンの押下
ビット4：右ボタンの開放
各ビット1のときにコールする
複数指定可能

ES：DX←ユーザー定義サブルーチンのアドレス

出　力　なし

解　説

ユーザーが作成したサブルーチンをマウスドライバがコールする条件とそのアドレスを設定します．このコールは次の手順で行われます．まずマウス割り込みによって制御がマウスドライバに移ります．次に，マウスドライバは指定されたコール条件のうち1つでも条件を満たせばサブルーチンをコールします．ここで，マウスドライバからのコールはユーザー定義サブルーチンがFAR型プロシージャとして行われます．したがってユーザーはサブルーチンをFAR型のプロシージャとして作成してください．

また，マウスドライバがサブルーチンをコールする際には次のような情報をレジスタに格納してコールします．よって，ユーザー定義サブルーチン内ではこれらのレジスタを参照することによりマウスの情報を利用できます．

AX→コールの原因となった現象

1：カーソルの位置変化
2：左ボタンの押下
4：左ボタンの開放
8：右ボタンの押下
16：右ボタンの開放

BL→左ボタンの状態

0：開放されている
−1：押されている

BH→右ボタンの状態

0：開放されている
−1：押されている

CX→カーソル位置の水平座標

DX→カーソル位置の垂直座標

13 ミッキー／ドット比設定 ハイレゾ

割り込み INT 33H

入力 AX←0FH

CX←水平方向のミッキー／ドット比

DX←垂直方向のミッキー／ドット比

出力 なし

解説 マウス本体の移動距離（ミッキー単位）とそれに対応する画面上のマウスカーソルの移動距離（ドット単位）の比を設定します．この設定はマウスカーソルを8ドット移動させるのに要するマウス本体の移動距離（ミッキー／8ドット，1ミッキーは約0.25mm）を単位として設定します（水平，垂直方向はそれぞれ個別に設定可能）．

したがってこの値を大きくすればマウスの感度は低くなり，値を小さくすればマウスの感度が高くなります．また，負の値を入れることにより，マウスの移動方向を反転させることもできます．

14 マウスカーソル表示画面，表示パターン設定 ハイレゾ

割り込み INT 33H

入力 AX←10H

BX←マウスカーソルの描画状態

Df	De	Dd	Dc	Db	Da	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

D0：プレーン0表示　1：表示する　0：表示しない

D1：プレーン1表示　1：表示する　0：表示しない

D2：プレーン2表示　1：表示する　0：表示しない

D3：プレーン3表示　1：表示する　0：表示しない

D4：カーソルの形状4個を指定

D5：カーソルの形状1個を指定

D7：AND，XOR指定　1：AND，XOR　0：XORのみ

出力 なし

解　説　カーソルの表示画面，表示パターンを設定します．いくつかのパターンについて設定例を以下に示します．

●1個のカーソル形状データで1画面（プレーン0）にXORのみをとる場合

	Df	De	Dd	Dc	Db	Da	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
BX	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1

●1個のカーソル形状データで4画面にAND，XORをとる場合

	Df	De	Dd	Dc	Db	Da	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
BX	0	0	0	0	0	0	0	0	1	0	1	0	1	1	1	1

●4画面異なったカーソル形状データでXORのみをとる場合

	Df	De	Dd	Dc	Db	Da	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
BX	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

●4画面に異なったカーソル形状データでAND，XORをとる場合

	Df	De	Dd	Dc	Db	Da	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
BX	0	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1

1個のカーソル形状データを2つの画面だけに表示することはできません．機能コード09H（マウスカーソル形状の設定）で設定されるカーソル形状データのパターンはこのコマンドで指定するパターンに一致するように設定しなければなりません．

15　マウス割り込みアドレスの設定

割り込み　INT　33H

入　力　AX←11H

BX←マウスBIOS作業領域のセグメントベース

CX←割り込み間隔　1～FFH（8.3msにつき1）

出　力　なし

解　説　マウス割り込みの割り込みアドレス等を設定します．マウスを使用する際には，最初に必ずこのコマンドを実行しなければなりません．作業内容は以下のとおりです．

・割り込みベクタテーブルにマウス割り込み処理ルーチンのアドレス設定
・割り込みコントローラ8259の割り込み許可ビットのマスク解除
・マウスBIOS作業領域のセグメントベースの設定
・インターフェースボートからの割り込み間隔（インターフェースボードからの割り込みを何回に1回受け付けるか）の設定．

16 マウス割り込みの許可

割り込み INT 33H

入　　力 AX←12H

BX←マウス割り込み（8255Aの割り込み）制御

0：割り込み開始

1：割り込み停止

出　　力 なし

解　　説 マウス割り込みを停止，開始させます．

§2-12 プリンタ

98には，パラレルI/O用LSIとして，8255（または相当品）が内蔵されています．このパラレルI/Oインターフェースはプリンタ用のインターフェースとして利用されています．また，このインターフェースはセントロニクスインターフェース準拠になっています．このインターフェースを通して，プリンタにデータを送り，文字の印刷などを行います．ただし，ノーマルモードとハイレゾモードでは機能が違います．また，プリンタ用のインターフェースのコネクタも違います．

ノーマルモードでは，簡易型のセントロニクスインターフェースとなっています．この場合，データ信号線以外は，BUSY（プリンタがデータ受信不可能であることを示す信号）とPSTB（プリンタにデータを渡すときのタイミング信号）しかありません．よって，プリンタの現在の状態や，紙切れなどの信号はプリンタから受け取ることが出来ず，かなり貧弱なインターフェースとなっています．これらの常態を検出するには，特別な手法が必要となります．

ハイレゾモードでは，フルセントロニクスに対応しています．ノーマルモードのセントロニクスインターフェースと比べるとかなり機能が豊富です．こちらには，プリンタの紙切れ，オンライン，オフライン状態，エラーなど多くのプリンタの常態を確認することができます．

■2-12-1 プリンタのI／Oポート

プリンタにデータを出力するためのBIOSが用意されていますが，プリンタインターフェースに関しては，直接I/Oポートを操作して出力することもさほど難しくありません．ノーマルモードで，プリンタの常態を読み取る時など，ちょっとしたテクニックを使う時には，有用でしょう．

プリンタのコントロールのほかに，パラレルI／Oよりシステムの各情報を取得することができます．使用しているマシンのシステムクロックの取得などは，このI／Oにより行います．

以下にノーマルモードおよびハイレゾモードのI／Oポート一覧を示します．

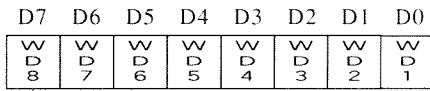
■プリンタのI／Oポート一覧（ノーマルモード）

リード／ライト	I／O アドレス	命令（機能）	D7	D6	D5	D4	D3	D2	D1	D0
リード	40H	リードデータ（書き込んだデータのチェック）	WD8	WD7	WD6	WD5	WD4	WD3	WD2	WD1
	42H	リードシグナル1（プリンタの状態の読込）	TYP1	TYP0	MOD	LCD	HGC	−BSY	CPUT	VF
	44H	リードシグナル2（ストローブ状態の読込）	−PSTB	*	*	*	IR8	*	RST278	*
ライト	37H	ライトポートC（PSTB信号Enable設定）	0	0	0	0	0	1	0	1／0
	40H	ライトデータ（プリンタにデータ送信）	WD8	WD7	WD6	WD5	WD4	WD3	WD2	WD1
	44H	ライトシグナル2（PSTB,IR8,RST信号制御）	−PSTB	0	0	0	IR8	0	RRT278	0
	46H	ライトモード（8255動作モード設定）	1	0	0	0	0		1	0
	46H	ライトシグナル1（287のリセット）	0	0	0	0	0	1	0	1／0
	46H	ライトシグナル1（IR8のON･OFF）	0	0	0	0	1	1	0	1／0
	46H	ライトシグナル1（PSTBのON･OFF）	0	0	0	0	1	1	1	1／0

■プリンタのI／Oポート一覧（ハイレゾモード）

リード／ライト	I／O アドレス	命令（機能）	D7	D6	D5	D4	D3	D2	D1	D0
リード	40H	リードデータ （書き込んだデータのチェック）	WD8	WD7	WD6	WD5	WD4	WD3	WD2	WD1
	42H	リードシグナル1 （プリンタの状態の読込）	$\overline{\text{SEL}}$	$\overline{\text{FAULT}}$	$\overline{\text{PE}}$	$\overline{\text{DC5V}}$	$\overline{\text{IBSY}}$	$\overline{\text{BSY}}$	$\overline{\text{ACKR}}$	$\overline{\text{ACK}}$
	44H	リードシグナル2 （ストローブ状態の読込）	$\overline{\text{OBF}}$	$\overline{\text{ACK}}$	*	*	IR14	$\overline{\text{PSTB}}$	$\overline{\text{RST278}}$	$\overline{\text{IPRIM}}$
ライト	40H 46H	ライトデータ （プリンタにデータ送信） ライトシグナル1 （Input PrimeのON/OFF）	W 0	W D 0	W D 0	W D 0	W D 0	W D 0	W D 0	W D 1 0 0
	46H	ライトシグナル1 （287のリセット）	0	0	0	0	0	0	1	1／0
	46H	ライトシグナル1 （PSTBのON/OFF）	0	0	0	0	0	1	0	1／0
	46H	ライトシグナル1 （INTEのON/OFF）	0	0	0	0	1	1	0	1／0
	46H	ライトシグナル1 （OBFのON/OFF）	0	0	0	0	1	1	1	1／0

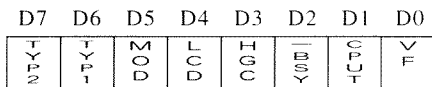
1 リードデータ /プリンタI/Oポート

アドレス **40H**

ライトデータで書き込んだデータがそのまま入ります．この機能はそれほど有用性はありません．

2 リードシグナル1 /プリンタI/Oポート

アドレス **42H**

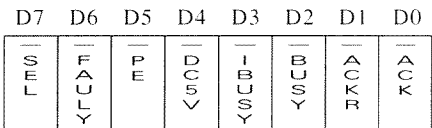
ノーマル

D7	D6	D5	D4	D3	D2	D1	D0
TYP2	TYP1	MOD	LCD	HGC	$\overline{BSY}$	CPUT	VF

- D0 VF　0：PC-9801VF以外　1：PC-9801VF
- D1 CPUT　0：CPUは80286以上　1：CPUはV30
- D2 $\overline{BSY}$　0：プリンタがデータ受信不可　1：プリンタがデータ受信可
- D3 HGC　0：拡張機能使用　1：拡張機能未使用
- D4 LCD　0：プラズマディスプレイ使用　1：プラズマディスプレイ未使用
- D5 MOD　0：システムクロック10MHz系　1：システムクロック8MHz系
- D7, D6 TYP2, TYP1

TYP1	TYP0	
0	0	：PC-9801
0	1	：未定義
1	0	：PC-9801/U2以外
1	1	：PC-9801U2

PC-9801／E／F／Mでは，対応していないビットがあります．

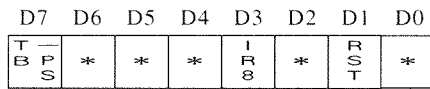
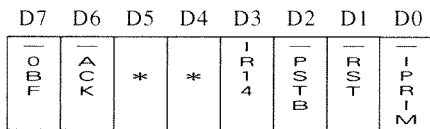
ハイレゾ

D7	D6	D5	D4	D3	D2	D1	D0
$\overline{SEL}$	$\overline{FAULY}$	$\overline{PE}$	$\overline{DC5V}$	$\overline{IBUSY}$	$\overline{BUSY}$	$\overline{ACKR}$	$\overline{ACK}$

それぞれのビットは，プリンタから送られて来る信号を意味します．しかしながら，使用するプリンタによって(メーカの違い)，それぞれのビットの意味が異なります．プリンタの説明書などで確認してください．

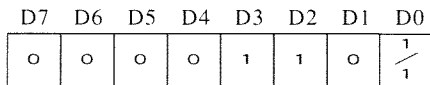
一般に，NECのプリンタでは，98で動作するよう作られていますので，ここに示した通りの信号が返ってきます．一方，EPSONなどのプリンタでは，世界的な標準である，IBM-PCにあわせた信号線になっています．ここに微妙な違いがあり，ハイレゾモードでは取得したプリンタの信号線の状態の意味が違ってしまいます．プログラムを作る上では注意してください．

なお，この問題が発生するのは，ハイレゾモード時および，H98や最近のMATEシリーズなどのフルセントロニクスインターフェースを利用した場合のみです．

3 リードシグナル2 ／プリンタI/Oポート

アドレス　44H

ライトシグナル2で書き込んだデータがそのまま読みだされます．

IR14および，RST以外は，プリンタの信号線の状態です．

4 ライトポートC／プリンタI/Oポート

アドレス　37H

このbitを1にすると，PSTBのON／OFFにかかわらず，常にPSTB信号が出力されません．パラレルI／OのLSIの初期化時にPSTBが出力されることがあるので，それを防ぐために，初期化時にこのbitを1にして，PSTBが出力されないようにします．

5 ライトポート /プリンタI/Oポート

アドレス 40H

プリンタに出力するデータを書き込みます．

6 ライトシグナル2 /プリンタI/Oポート

アドレス 44H

NDPとは，287，387などのコプロセッサのことです．

IR8を1にすることにより，8259（割り込みコントロールLSI）（詳しくは，「1-5. 割り込み」を参照）に割り込み信号が出力されます．プリンタ制御回路からの割り込みは，このビットを1にすることにより発生するわけですが，ソフトウェアにより発生させるわけですから，この割り込みはまず使い道はないでしょう．

PSTB信号は，プリンタにデータを送信する際に，一瞬だけ0：アクティブにします．ライトデータによりデータを出力しただけでは，プリンタはデータを受け取りません．プリンタ側はいつデータを受け取るかわからないからです．そこでデータを受け取るときを知らせるのが，PSTB信号です．正確には，ライトデータでデータを出力した後，1μ秒以上後にPSTBをアクティブにし，PSTBを1μ秒以上アクティブに保ち，またインアクティブにします．これが，1バイトプリンタに送るための一連の処理です．

NDPとは，287，387などのコプロセッサのことです．

7 ライトモード／プリンタI/Oポート

アドレス **46H**

ノーマル

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	1	0

プリンタインターフェースに使われているLSI，8255のモードを設定します．プリンタインターフェースを利用可能状態にするには，上記のようなデータを出力しなくてはなりません．

ハイレゾ

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	0	0	0	1	0

プリンタインターフェースに使われているLSI，8255のモードを設定します．プリンタインターフェースを利用可能状態にするには，上記のようなデータを出力しなくてはなりません．

8 ライトシグナル1／プリンタI/Oポート

アドレス **46H**

ノーマル・ハイレゾ ●287(387)のリセット

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	1	1/0

D0：
0：NDPをリセットしない
1：NDPをリセットする
（ともにCPUリセット時の動作設定）

●IR8のON・OFF

0：通常状態
1：8259へ割り込み信号発生

●PSTBのON・OFF

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	1	1	1/0

0：PSTBアクティブ
1：PSTBインアクティブ

ライトシグナル1は，ライトシグナル2の各ビットの設定を個別に行うものです．PSTB，IR8，RSTのすべてを設定するには，ライトシグナル2が便利であり，それぞれ1つだけ設定を変えるときにはライトシグナル1が便利であるといえます．

ハイレゾ

●Input PrimeのON・OFF

0：Input Primeアクティブ
1：Input Primeインアクティブ

●PSTBのON・OFF

0：PSTBアクティブ
1：PSTBインアクティブ

●INTEのON・OFF

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	1	0	1/0

0：INTE 禁止
1：INTE 許可

●OBFのON・OFF

0：OBFアクティブ
1：OBFインアクティブ

ライトシグナル1は，ライトシグナル2の各ビットの設定を個別に行うものです．PSTB，IR8，RSTのすべてを設定するには，ライトシグナル2が便利であり，それぞれ1つだけ設定を変えるときにはライトシグナル1が便利であるといえます．

■サンプルプログラム

```
/*  システムの状態を取得し，表示します．
    (ノーマルモード用)  */
#include <stdio.h>
#include <dos.h>

void main(void)
{
      unsigned char data;

      data = inportb(0x42);   /* リードシグナル1 */
      if(data & 2) {
            printf("CPUはV30です\n");
      } else {
            printf("CPUは286以上です\n");
      }
      if(data & 4) {
            printf("プリンタはデータ受信可能です\n");
      } else {
            printf("プリンタはデータ受信不可能です\n");
      }
      if(data & 0x20) {
            printf("システムクロックは8MHz系です\n");
      } else {
            printf("システムクロックは10MHz系です\n");
      }
}
```

■2-12-2 プリンタのBIOS

98には，プリンタを簡単に制御するためのBIOSが用意されています．これを用いることによって，非常に簡単にプリンタにデータを出力することができます．

ハイレゾモードでは，BIOSで用紙切れや，プリンタの電源OFFなどの情報まで検出することが可能になっています．

■プリンタBIOS一覧（INT 1AH）

機能コード	機能		ノーマル	ハイレゾ
10H	プリンタBIOS初期化		○	○
11H	データを出力	(waitモード)	○	○
12H	ステータスの取得		○	○
14H	データの出力	(no waitモード)	×	○
15H	データの出力	(チェックレスモード)	×	○
16H	初期化	(wait時間設定)	×	○
30H	データの出力	(複数バイト)	○	○

※ハイレゾモードのプリンタBIOSでは，どのファンクションコールでも出力されるステータス情報は共通です．ステータスの詳しい説明は，p.393のハイレゾモードのステータスを参照してください．

1 プリンタ　BIOS初期化

割り込み INT 1AH

入　力 AH←10H

出　力 ◆ノーマルモードの場合

AH→00H：プリンタBUSY
　　01H：データ送信可

◆ハイレゾモードの場合

ハイレゾモードのステータス（p.393）を参照

解　説 プリンタBIOSの初期化を行います．プリンタBIOSを利用するときは，必ずこのファンクションコールを一番最初に1度実行してください．

なお，ハイレゾモードでは，このファンクションコールを実行すると，プリンタBUSYのウエイト時間が，4秒に設定されます．

サンプル

```
/*  プリンタBIOSを初期化し，現在のプリンタの状態を表示します．
    （ノーマルモード用）  */
#include  <stdio.h>
#include  <dos.h>

union  REGS  inregs,  outregs;

void  main(void)
{
        inregs.h.ah  =  0x10;
        int86(0x1a,  &inregs,  &outregs);
        if(outregs.h.ah  !=  0)  {
                printf("プリンタにデータ送信可能です．\n");
        } else  {
                printf("プリンタがBUSYです．\n");
        }
}
```

2 データの出力（waitモード）

割り込み INT　1AH

入　力 AH←11H

AL←出力する1バイトデータ

出　力 ◆ノーマルモードの場合

AH→00H：プリンタBUSY

01H：データ送信可

02H：タイムアウト，データ未出力

◆ハイレゾモードの場合

ハイレゾモードのステータス（p.393）を参照

解　説

プリンタに送信可能状態になるまでループして待ち，送信可能になった時点で，1バイトデータをプリンタに送信します．一定時間経過しても送信可能状態にならない場合は，タイムアウトとなり，データを送信せずに終了します．

ノーマルモードの場合，プリンタのディセレクト，用紙切れなどは判断できないので，このような場合は，タイムアウトとして終了します．また，プリンタの電源のON・OFF状態も判断できません．プリンタの電源がOFFの場合は，常にデータ送信可能状態と判断されますので，注意が必要です（この場合，BIOSは正常にプリンタにデータを送ったと判断します）．

ハイレゾモードの場合は，ノーマルモードと違い，ステータスにオフライン，用紙切れ，電源の状態などが返ってきます．

サンプル

```
/*  プリンタに#define で定義した値（1バイト）を出力します
    このプログラムを実行する前に，プリンタBIOSを初期化しておく必要があります
   （ノーマルモード用）     */
#include  <stdio.h>
#include  <dos.h>

#define  OUTDATA  0x0d        /*  出力データ（この場合は改行コード）  */

union  REGS  inregs,  outregs;

void  main(void)
{
        inregs.h.ah  =  0x11;
        inregs.h.al  =  OUTDATA;
        int86(0x1a,  &inregs,  &outregs);
        if(outregs.h.ah  ==  0x02)  {
                printf("タイムアウト  データ未送信です.\n");
        }  else  {
                printf("データ送信が終了しました.\n");
        }
}
```

3 ステータスの取得

割り込み INT　1AH

入　力 AH←12H

出　力 ◆ノーマルモードの場合

AH→00H：プリンタBUSY

01H：データ送信可

◆ハイレゾモードの場合

ハイレゾモードのステータス（p.393）を参照

解　説　現在のステータスを取得し，[出力]で示したような値をセットし，戻ります．

サンプル

```
/*  現在のプリンタの状態を取得して，表示します
    このプログラムを実行する前に，プリンタBIOSを初期化しておく必要があります
   (ノーマルモード用)  */
#include <stdio.h>
#include <dos.h>

union REGS inregs, outregs;

void main(void)
{
        inregs.h.ah = 0x12;
        int86(0x1a, &inregs, &outregs);
        if(outregs.h.ah == 0) {
                printf("プリンタはBUSYです. \n");
        } else {
                printf("プリンタにデータ送信可能です. \n");
        }
}
```

4　データの出力（no　waitモード）　ハイレゾ

割り込み　INT　1AH

入　力　AH←14H

AL←出力する1バイトデータ

出　力　ハイレゾモードのステータスを参照

解　説　このファンクションは，ハイレゾモードでのみサポートされます．

プリンタにデータを送信可能であれば，1バイトデータを送信します．送信不可能な場合は，AX←8208H，INT 1FHを行い，いったんCPUを開放します．INT 1FHから戻ってきた時点で，再度プリンタに送信可能かチェックし，可能であれば，送信します．不可能であれば，再度同じ様に，INT 1FHを行います．

このように，送信可能になるまで行いますが，一定時間送信可能にならなければ，タイムアウトとなり，戻ります．

5 データの出力（チェックレスモード） ハイレゾ

割り込み INT 1AH

入　　力 AH←15H

AL←出力する1バイトデータ

出　　力 ハイレゾモードのステータスを参照

解　　説 このファンクションは，ハイレゾモードでのみサポートされます．

プリンタのステータスをチェックせずに，1バイトデータを送信します．

6 初期化(wait時間設定) ハイレゾ

割り込み INT 1AH

入　　力 AH←16H

CX←ウエイト時間（10ms単位）

0を与えると，BUSYでなくなるまで待ちます．

出　　力 ハイレゾモードのステータスを参照

解　　説 このファンクションは，ハイレゾモードでのみサポートされます．

プリンタ用のLSIの初期化，ステータスの初期化を行います．また，プリンタがBUSYである場合のウエイト時間を設定します．

データ送信のときに，CXレジスタで設定した時間だけ，プリンタがBUSYであっても，データの出力を待ちます．この時間以上たってBUSYのままなら，送信をせずに戻ります．与えるウエイト時間は，10ms単位であることに注意してください．最大値である，FFFFHを与えると，655350ms待ちます．

7 データの出力(複数バイト)

割り込み INT 1AH

入　　力 AH←30H

CX←データの長さ

ES←出力データのある先頭アドレス（セグメント）

BX←出力データのある先頭アドレス（オフセット）

出　力

◆ノーマルモードの場合

AH→00H：正常終了

02H：タイムアウト

◆ハイレゾモードの場合

ハイレゾモードのステータス（p.393）を参照

◆ノーマル，ハイレゾモード共通の出力データ

BX→タイムアウト時の出力データアドレス（オフセット）

CX→タイムアウト時に送信していないデータのバイト数

解　説

CXで指定したバイト数のデータをプリンタに出力します．なお，出力データのバッファは，セグメントの境界を超えないようにしなくてはいけません．

●ハイレゾモードのステータス

ハイレゾモードのプリンタBIOSでは，返されるステータスは各ファンクションで共通になっています．以下のようなステータスが返されます．

AH→00H：プリンタBUSY

01H：データ送信が正しく終了，またはデータ送信可能

02H：タイムアウト

03H：プリンタがオフライン（セレクト状態でない）

04H：用紙切れ

05H：プリンタの電源OFF，または未接続

AL→プリンタのステータス信号の状態

D7	D6	D5	D4	D3	D2	D1	D0
SELECT	FAULT	PE	DC5V	I.BUSY	BUSY	*	ACK

●プリンタにデータを出力する上での注意

I／Oの直接制御，BIOSを使ってのデータ出力に共通することですが，プリンタに対しては，プリンタ制御用のデータおよび，印刷文字のデータが出力されるわけですが，プリンタの制御コードは各プリンタにより異なります．代表的なものでは，以下のようなものがあります．

- ESC／P
- PR201
- LIPS2，3
- ESC／Page

利用するプリンタにあった制御コードを使います．詳しくはプリンタメーカーで用意している，プリンタのマニュアルや，制御コードマニュアルなどを参照してください．なお，ESC／Pおよび，PR201に関しては，基本的な制御コードを「第5章　資料編」（5－11．プリンタ制御コード表）に掲載しています．

また，印刷文字データですが漢字を印刷するときには，注意が必要です．98ではテキストファイルなど，一般に漢字コードはSHIFT　JISを用いていますが，プリンタが受け付ける漢字コードは，JISコードです．JISコードに変換しないと漢字が印刷できません．

■サンプルプログラム

テキストファイルを読み込み，プリンタに印字するプログラムです．使用するプリンタにあわせて，変数PRINTERを変更してください．

```
/*
    テキストファイルを読み込んで印刷します
    （文字間隔を小さめにし，ANK:漢字 = 1:2 の印刷を行います）

   プリンタは，ESC/P,PR201どちらでも可.
   ただし，使用するプリンタに合わせて PRINTER の定義を変更
*/

#include  <stdio.h>
#include  <stdlib.h>
#include  <dos.h>
#include  <string.h>
#include  <jctype.h>
#include  <jstring.h>

#define  PRINTER  0              /*  0:ESC/P   1:PR201  */
#define  STEP  1                 /*  改行幅  */

#if  0  ==  PRINTER

#define  INIT  "\x1b""@"
#define  KANJIMODE  "\x1c""&""\x1c""S""\x03""\x03"      /*漢字モード指定&
                                                          漢字ドットスペース3+3  */
#define  ANKMODE  "\x1b""M""\x1c""."                    /*ANKモード(12CPI)指定  */

#else

#define  INIT  "\x1b""c1"
#define  KANJIMODE  "\x1b""K""\x1c""C"                  /*  漢字モード指定  */
#define  ANKMODE  "\x1b""E"                             /*  ANKモード(エリート)指定  */

#endif

char  prt_init(void);
void  prt_send(char);
void  err(int);
void  line_send(char  *);
void  line_send2(char  *);
```

```
union REGS inregs, outregs;

void main(void)
{
	FILE *fp;
	char filename[80];
	unsigned char buf[512];
	int l;

	printf("印刷するファイル名を入力してください. \n");
	scanf("%s", filename);
	if((fp = fopen(filename, "r")) == NULL) {
		err(0);
	}
	if(prt_init() == 0) {
		err(1);
	}
	line_send2(INIT);
	line_send2(ANKMODE);
	while(feof(fp) == 0) {
		fgets(buf, 512, fp);
		l = strlen(buf);
		buf[l - 1] = '\0';
		line_send(buf);
	}
	prt_send(0x0c);                         /* 改頁コード送信 */
	fclose(fp);
}

char prt_init(void)                             /* プリンタBIOS初期化 */
{
	inregs.h.ah = 0x10;                     /* 初期化ファンクション */
	int86(0x1a, &inregs, &outregs);
	return outregs.h.ah;
}

void prt_send(char data)
{
	inregs.h.ah = 0x11;                     /* データ出力ファンクション */
	inregs.h.al = data;
	int86(0x1a, &inregs, &outregs);
	if(outregs.h.ah == 0x02) {              /* タイムアウトしたときは, */
		printf("time out!!\n");         /* データはプリンタに送信 */
	}                                       /* されません */
}

void err(int no)
{
	switch(no) {
	case 0:
		printf("ファイルオープンエラー\n");
		break;
	case 1:
		printf("プリンタの準備が出来ていません\n");
```

```
                break;
            }
            exit(-1);
}

void line_send(char *buf) /* プリンタに1ラインデータを送る */
/* buf : 印字データ */
{
        unsigned short c;
        int l, i, kanji;

        kanji = 0;
        l = strlen(buf);
        for(i = 0; i <= l - 1; i++) {
                if(kanji == 0 && (nthctype(buf, i) == 1 ||
                nthctype(buf, i + 1) == 2)) {
                        line_send2(KANJIMODE);
                        kanji = 1;
                }
               if(kanji == 1 && nthctype(buf,i)== 0 && nthctype(buf,i+1) != 2) {
                        line_send2(ANKMODE);
                        kanji = 0;
                }
                if(kanji == 1) {
                   c = (*(buf + i) & 0x00ff) * 0x100 + (*(buf + i + 1) & 0x00ff);
                        i++;
                        c = jmstojis(c);            /* 漢字コード変換(SJIS->JIS) */
                        prt_send(c / 0x100);
                        prt_send(c & 0x00ff);
                }
                if(kanji == 0) {
                        prt_send(*(buf + i));
                }
        }
        line_send2(ANKMODE);
        prt_send(0x0d);
        prt_send(0x0a);
}

void line_send2(char *buf) /* プリンタに1ラインデータを送る(漢字含まない) */
{
        int l, i;

        l = strlen(buf);
        for(i = 0; i <= l - 1; i++) {
                prt_send(*(buf + i));
        }
}
```

§ 2-13 FM音源

FM音源にもBIOSは存在し，サウンドボード上にROMの形で供給されています．このBIOS ROMのアドレスは，SCSI I/FのBIOS ROMアドレスやハードウェアEMSのページアドレスと干渉してしまうため，通常，これらの機器とFM音源を同時に使うことは困難になります．このような場合，EMS上にROMの内容をコピーして使うなどの工夫をすればよいのですが，最近ではBIOSを使用することなく直接制御を行って，サウンド機能を実現するものが多くなっています．ここでは，直接制御する方法を説明することにします．

PC-9821が発売されるまで，PC-9801シリーズは，FM音源用LSIとしてYM-2203(OPN)が搭載されていました，また，FM音源が搭載されていない機種も，PC-9801-26/Kを増設することにより使用が可能となっています．新しいFM音源(YM-2608 OPNA 搭載)はOPNの上位互換になってますので，拡張された部分を使わなければ従来と同様に使用することができるようです．したがってこれから説明することは，あくまでOPNを搭載した従来型のFM音源で有効であることをあらかじめお断りしておきます．

■2-13-1 FM音源のコントロール

FM音源で音を鳴らすには，次の手順を踏む必要があります．

①音色の設定
②鳴らすチャネルの設定
③音を出す
④必要な時間だけ待つ
⑤音をとめる

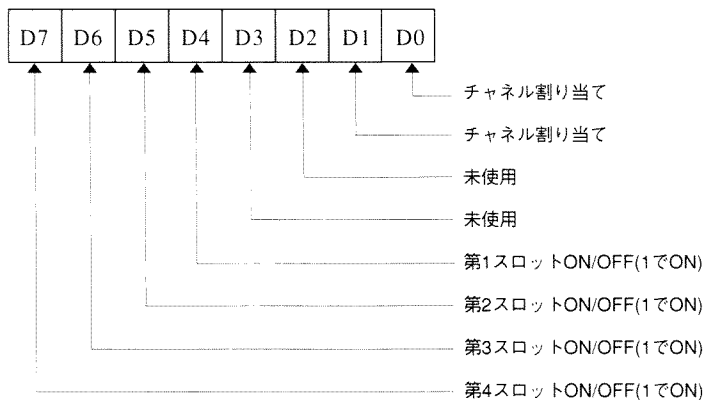
①音色の設定

音色は，ここでは説明しませんが各種音色エディタでいろいろ作ることができます．

②チャネルの設定

OPNは3チャネルありますので，そのチャネルを指定します．

③音を出す

OPNにKeyOnを送り，音を出します．OPN Address 28Hが，音を出すチャネルとスロットをコントロールします．ビット割り当ては，次のようになってます．

チャネル割り当ては，2bitでコントロールします．各チャネルとの対応は以下のようになっています．

D1	D2	チャネル
0	0	チャネル1
0	1	チャネル2
1	0	チャネル3

④音の長さだけ待つ

音の長さの分だけ待ちます．通常は割り込みを使用して行います．

⑤音を止める

OPNにKeyOFFを送り，音をとめます．

これらの操作はI/Oポートを制御することによって実現します．これらの具体的な作法は，サンプルプログラム（p.415）を参照してください．

●I/Oポート

FM音源ボードに割り当てられているI/Oポートは表のとおりです．

■FM音源のI/Oポート

リード／ライト	I／O アドレス	命令（機能）	データ							
ライト	0188H	ライトアドレス	A7	A6	A5	A4	A3	A2	A1	A0
	018AH	ライトデータ	D7	D6	D5	D4	D3	D2	D1	D0
リード	0188H	リードステータス	BUSY	・	・	・	・	・	FlagA	FlagB
	018AH	リードデータ	D7	D6	D5	D4	D3	D2	D1	D0

※注意 Read Dataは内部アドレスが00から0fhに限る

C言語から制御する場合はoutportbによりアクセスすることができます．

●FM音源の内部レジスタ

OPNの内部レジスタを表に示します．この内部レジスタを設定することによってOPNを制御します．

表2-40　OPN読み込みデータ

Read Data Address FM音源の内部アドレス	データ D7	D6	D5	D4	D3	D2	D1	D0	コメント 説明
00H～0FH	BUSY	·	·	·	·	·	FlagA	FlagB	ステータス

表2-41　OPNの内部レジスタ (Part-1)FM音源部（その1）

Write Data Address FM音源の内部アドレス	データ D7	D6	D5	D4	D3	D2	D1	D0	コメント 説明
21H	Test								LISのTEST DATA
24H	TIMER A								TIMER A の上位8bit
25H	·	·	·	·	·	·	TIMERA	TIMERA	TIMER A の下位2bit
26H	TIMER B								TIMER BのDATA
27H	MODE	MODE	RESETB	RESETA	ENABLEB	ENABLEA	LOAD B	LOAD A	TIMER A/Bの制御と3chのモード
28H	←SLOT→				·	·	←CHANNEL→		Key-ON/OFF
2DH	·	·	·	·	·	·	·	·	プリスケーラをset
2EH	·	·	·	·	·	·	·	·	1/3,1/6分周の選択
2FH	·	·	·	·	·	·	·	·	分周器を1/2にセット

表2-42　OPNの内部レジスタ (Part-2)FM音源部（その2）

Write Data Address FM音源の内部アドレス	データ								コメント
	D7	D6	D5	D4	D3	D2	D1	D0	説明
3 0 h : 3 E H	·	Detune			Multiple				Detune/Multiple (33h,37h,3bhの Addressはなし)
4 0 H : 4 E H	·	Total Level							ToTal Level (43h,47h,4bhの Addressはなし)
5 0 h : 5 E H	Key Scal		·	Attack Rate					KeyScale/AttackRate (53h,57h,5bh の Addressはなし)
6 0 h : 6 E H	·	·	·	Decay Rate					Decay Rate (63h,67h,6bh のAddressはなし)
7 0 h : 7 E H	·	·	·	Sustain Rate					Sustain Rate (73h,77h,7bh のAddressはなし)
8 0 h : 8 E H	Sustain Level				Release Rate				SustainLevel/Release Rate (83h,87h,8bh のAddressはなし)
9 0 h : 9 E H	·	·	·	SSG-Type Envelop Contorol					SSG-Type Envelop Contorol(93h,97h,9bh のAddressはなし)
A 0 H A 1 H A 2 H	F-Numbers 1								F-Numberの下位8bit
A 4 H A 5 H A 6 H	·	·	BLOCK			F- Numbers 2			BLOCK/F-Numberの 上位3bit
A 8 H A 9 H A A H	3Ch-F-Numbers 1								3Ch-3Slot F-Numberの下位8bit
A C H A D H A E H	·	·	3Ch- BLOCK			3Ch-F- Numbers 2			3Ch-3Slot BLOCK/F-Numberの 上位3bit
B 0 H B 1 H B 2 H	·	·	Self Feedback			Connection			Self-Feedback /Connection

表2-43　OPNの内部レジスタ（その3）SSG音源部

Write Data Address FM音源の内部アドレス	データ D7	D6	D5	D4	D3	D2	D1	D0	コメント 説明
0 0 H	Fine Tune								Channel-A Tone Period下位8bit
0 1 H	·	·	·	·	Coarse Tune				Channel-A Tone Period上位4bit
0 2 H	Fine Tune								Channel-B Tone Period下位8bit
0 3 H	·	·	·	·	Coarse Tune				Channel-B Tone Period上位4bit
0 4 H	Fine Tune								Channel-C Tone Period下位8bit
0 5 H	·	·	·	·	Coarse Tune				Channel-C Tone Period上位4bit
0 6 H	·	·	·	Period Control					Noise Period
0 7 H	In/Out B	A	NoiseEnable C	B	A	ToneEnable C	B	A	ミキサーとI/Oのコントロール (0でON，1でOFF)
0 8 H	·	·	·	M	Level				Cannel A Amplitude とMode
0 9 H	·	·	·	M	Level				Cannel B Amplitude とMode
0 A H	Fine Tune								Cannel C Amplitude とMode
0 B H	Coarse Tune								Envelop Period 下位8ビット
0 C H	Contorol								Envelop Period 上位8ビット
0 D H	·	·	·	·	C	ATT	ALT	HLD	Envelop形状
0 E H	I/O Port A								I/O Port Data
0 F H	I/O Port A								I/O Port Data

1 ステータスの取得 /FM音源内部レジスタ

I/Oアドレス **018AH**

内部アドレス **00H～0FH**ですが，特に指定する必要は有りません．

解説　FM音源の状態を調べます．調べられる情報は表2-40（p.400）のOPN読み込みデータの通りです．

サンプル　FM音源がBUSYでなくなるまでWAITします

```
/* dos.hをincludeの事 */

#define READ_STATUS 0x0188

/* busy flag check */
do {
     result = inportb( READ_STATUS );
}while( result & 0x80 );
```

2 LSIのテストデータ /FM音源内部レジスタ

I/Oアドレス **018AH**

内部アドレス **21H**

解説　FM音源のテスト用アドレスです．常に0に保たなければなりません．

3 TIMER A / FM音源内部レジスタ

I/Oアドレス **018AH**

内部アドレス **24H，25H**

解説　Timer Aのプリセット値(動作周期の長さを決める値)を入力します．通常の分周比では，次の計算式によって動作周期を求める事ができます．

NA： 24h, 25hに設定する10bitの値

f： マスタクロック周波数(3.9936MHz)/分周数

T = 12×(1024 - NA)/f

T = 12×(1024-NA)/665.6(デフォルト設定)

4 TIMER Bのデータ ／FM音源内部レジスタ

I/Oアドレス 018AH

内部アドレス 26H

解説

Timer Bのプリセット値(動作周期の長さを決める値)を入力します．通常の分周比では，次の計算式によって動作周期を求める事ができます．

NA：26h に設定する 8bit の値

f ：マスタクロック周波数(3.9936MHz)/分周数

T = 192×(256 - NB)/f

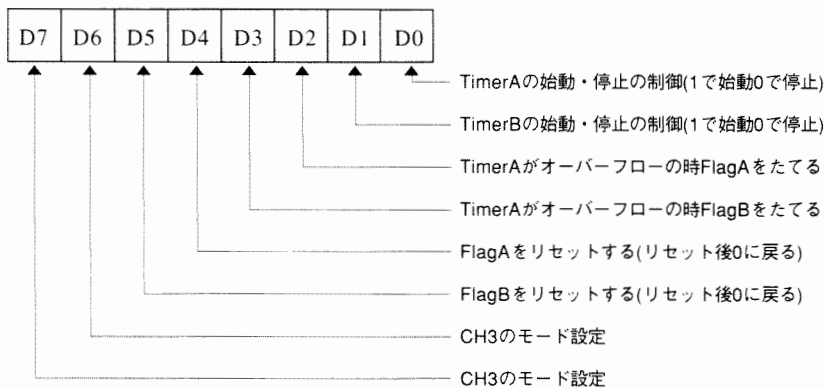
T = 192×(256-NB)/665.6(デフォルト設定)

5 TIMER A/Bの制御と3chのモード／FM音源内部レジスタ

I/Oアドレス 018AH

内部アドレス 27H

解説

タイマーA／Bの制御などを行います．

CH3のモード設定は，以下のように設定する事ができます．

MODE	D7	D6	内容
通常	0	0	チャネル3は通常の発音モードとなります．F-NumberはA2H,A6Hで設定できます
効果音	0	1	チャネル3のF-Numberは各スロット毎に設定できます (1SLOT:A9H,ADH/2SLOT:AAH,AEH/3SLOT:A8H,ACH)
音声合成	1	0	チャネル3のF-Numberは効果音の場合と同様ですが，MODEはCSM音声合成モードとなりKey ON/OFFがTimerAによって制御されます

6 KEY-ON／OFF／FM音源内部レジスタ

I/Oアドレス **018AH**

内部アドレス **28H**

解説　Key ON/OFFの制御を行います．上位4bitsがスロットの指定を，下位2bitがチャネルの指定を制御します．詳しい事は，p.397の③の音を出すの項を参考にしてください．

サンプル　チャネル1のすべてのスロットをKey ONにします．

```
#define READ_STATUS 0x0188
#define WRITE_ADDRESS 0x0188
#define WRITE_DATA 0x018a

      /* busy flag check */
      do {
            result = inportb( READ_STATUS );
      }while( result & 0x80 );

      /* OPNへアドレス出力 */
      outportb( WRITE_ADDRESS, 0x28 );

      /* busy flag check */
      do {
      result = inportb( READ_STATUS );
      }while( result & 0x80 );

      /* wait */
      outportb( 0x5f, 0 );
      outportb( 0x5f, 0 );
      outportb( 0x5f, 0 );
      outportb( 0x5f, 0 );
      outportb( 0x5f, 0 );

      outportb( 0x5f, 0 );
      outportb( 0x5f, 0 );
      outportb( 0x5f, 0 );
      outportb( 0x5f, 0 );
      outportb( 0x5f, 0 );

      outportb( 0x5f, 0 );
      outportb( 0x5f, 0 );
      outportb( 0x5f, 0 );
      outportb( 0x5f, 0 );
      outportb( 0x5f, 0 );

      outportb( 0x5f, 0 );
      outportb( 0x5f, 0 );
      outportb( 0x5f, 0 );
      outportb( 0x5f, 0 );
      outportb( 0x5f, 0 );
```

```
        /* OPNへデータ出力 */
        outportb( WRITE_DATA, 0xf0 );
```

7 プリスケーラをセット／FM音源内部レジスタ

I/Oアドレス 018AH

内部アドレス 2DH,2EH,2FH

解説　ここでは，FM,SSG各音源の基底周波数を制御します．この3つの内部アドレスにはデータビットはないので，単にアドレスの選択を行うだけで基底周波数を制御できます．

基底周波数と，アドレスの関係は次のようになっています．

2D	2E	2F	FM音源の分周数	SSG音源の分周数	OPNに入力できる最大周波数
--	--	ON	2	1	1.4MHz
ON	--	--	6	4	4.2MHz
ON	ON	--	3	2	2.1MHz

※ 2-13-3のプリスケーラの項で詳しい説明をおこなっています．

サンプル　FM音源の分周比を1/2にする(通常禁止されている動作です)

```
#define READ_STATUS 0x0188
#define WRITE_ADDRESS 0x0188
#define WRITE_DATA 0x018a

        /* busy flag check */
        do {
                result = inportb( READ_STATUS );
        }while( result & 0x80 );

        /* OPNへアドレス出力 */
        outportb( WRITE_ADDRESS, 0x2f );

/* 次のFM音源操作のためのwaitです */

        /* busy flag check */
        do {
        result = inportb( READ_STATUS );
        }while( result & 0x80 );

        /* wait */
        outportb( 0x5f, 0 );
        outportb( 0x5f, 0 );
        outportb( 0x5f, 0 );
        outportb( 0x5f, 0 );
        outportb( 0x5f, 0 );

        outportb( 0x5f, 0 );
        outportb( 0x5f, 0 );
```

```
        outportb( 0x5f, 0 );
        outportb( 0x5f, 0 );
        outportb( 0x5f, 0 );

        outportb( 0x5f, 0 );
        outportb( 0x5f, 0 );
        outportb( 0x5f, 0 );
        outportb( 0x5f, 0 );
        outportb( 0x5f, 0 );

        outportb( 0x5f, 0 );
        outportb( 0x5f, 0 );
        outportb( 0x5f, 0 );
        outportb( 0x5f, 0 );
        outportb( 0x5f, 0 );
```

8 Detune/Multiple ／FM音源内部レジスタ

I/Oアドレス　**018AH**

内部アドレス　**30H〜3EH**

解説

MultipleはF-NumberとDetuneで得られる位相情報に対して，次の表であらわされる倍率を与えます．Detuneは，F-Numberにわずかな周波数ずれを与える情報です．

Multiple	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
倍率	1/2	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

サンプル

サンプルプログラム（p.415）を参照してください．

9 Total Level ／ FM音源内部レジスタ

I/Oアドレス　**018AH**

内部アドレス　**40H〜4EH**

解説

トータルレベルとは，エンベロープジェネレータの出力に対して，減衰量を加算し，変調度および音色の制御をするために使用されます．減数量は最少分解能を0.75dBとして，各bitの重み付けは表の通りです．

トータルレベルは，CSMモードを選択した場合3チャネルの音に対し，エンベロープのイニシャルレベルとなります．よって，エンベロープの途中でトータルレベルを変えてもエンベロープの減衰量は変化せず次のKey-ON時のレベルが変わります．

TotalLeve	D6	D5	D4	D3	D2	D1	D0
減衰量(dB)	48	24	12	6	3	1.5	0.75

サンプル

サンプルプログラム（p.415）を参照してください．

10 Key Scale/Attack Rate /FM音源内部レジスタ

I/Oアドレス 018AH

内部アドレス 50H～5EH

解説 Key Scaleは，エンベロープのRateを音程によって変化させるために設けられています．Key Scaling後のRateは次の式によって与えられます．

R ：入力したRate
Rks：Key Scaling量(表を参照)
Rate = 2×R + Rks
(ただしR=0の時はRate=0とする)

Attack Rateは，音の立ち上がりの時間を設定します．

表2-44 RateのKey Scale

Key Code																
KS	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
1	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
2	0	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7
3	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
KS	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	3
1	4	4	4	4	5	5	5	5	6	6	6	6	7	7	7	7
2	8	8	9	9	10	10	11	11	12	12	13	13	14	14	15	15
3	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

サンプル サンプルプログラム（p.415）を参照してください．

11 Decay Rate /FM音源内部レジスタ

I/Oアドレス 018AH

内部アドレス 60H～6EH

解説 Decay時のRateを設定します．

サンプル サンプルプログラム（p.415）を参照してください．

12 Sustain Rate ／ FM音源内部レジスタ

I/Oアドレス　018AH

内部アドレス　70H～7EH

解説　Sustain時のRateを設定します．

サンプル　サンプルプログラム（p.415）を参照してください．

13 Sustain Level/Release Rate／FM音源内部レジスタ

I/Oアドレス　018AH

内部アドレス　80H～8EH

解説　Sustain LevelはDecay Rateと，Sustain Rateの切り替え点を与えます．その重み付は表にある通りです．ただし，Sustain Levelのすべてのbitが1の時は減衰量は93dBになります．

Release RateはKey-offの時のRateを与えます．ただしRelease Rateは最下位bitが1に固定されているため他のRateと違い4bitsしか入力できません．

SustainLevel	D7	D6	D5	D4
減衰量(dB)	24	12	6	3

(D4～D7が1の時は減衰量は93dB)

サンプル　サンプルプログラム（p.415）を参照してください．

14 SSG-Type Envelop Control ／FM音源内部レジスタ

I/Oアドレス　018AH

内部アドレス　90H～9EH

解説　SSG-Type Envelop Contorolとは，エンベロープの形状をSSG音源と同様に与えるものです．データビットの最上位(D4)がこのタイプのエンベロープを選択するスイッチになっておりまして，残りの3bitsでエンベロープの選択が行われます．

サンプル　サンプルプログラム（p.415）を参照してください．

15 F-Number ／ FM音源内部レジスタ

I/Oアドレス 018AH

内部アドレス A0H A1H A2H／A4H A5H A6H

解説　このアドレスでF-Numberを設定します．データをセットする場合の注意点としては，まず最初にBlockとF-Numbersの上位3bits(F-Numbers2)を設定し，次にF-Numbersの上位8bits(F-Numbers1)を設定しなければなりません．内部レジスタへのloadingはF-Numbers1が設定された時ですから，順序が逆だと正しく入力されません．

では，F-Number/Blockは音程を決めるものです．次のようにして求める事ができます．まず，サンプリング周波数f SAMを求めます．与えるパラメータは，入力クロック周波数fM(3.9936MHz)と，プリスケーラコントロールで得られる分周数n(6)です．式は，

$$f\,SAM = fM/(12 \times n)$$

通常使用するばあい(デフォルト設定)では，次のようになります．

$$f\,SAM = 3.9936 \times 10^6/(12 \times 6) = 55467Hz$$

F-Numberは次の式であたえられます．

f Mus：発音したい周波数
b　：オクターブデータ(Block)

$$F\text{-}Number = (f\,Mus \times 2^{20}/f\,SAM)/2^{(b-1)}$$

例としてA4(440Hz)のF-Numberを求めてみましょう．

$$F\text{-}Number = (440 \times 2^{20}/55467)/2^{(4-1)} = 1040$$

サンプル　サンプルプログラム（p.415）を参照してください．

16 3Ch-3Slot F-Number ／FM音源内部レジスタ

I/Oアドレス 018AH

内部アドレス A8H A9H AAH／ACH ADH AEH

解説　このアドレスで設定されるF-NumbersとBlockは効果音モードやCSM音声合成モードで使用されるデータです．セットの方法は前と同様です．

ここでセットされるデータとスロットの関係は次のようになってます

SLOT	Address
1	A9H,ADH
2	AAH,AEH
3	A8H,ACH
4	A2H,A6H

サンプル　サンプルプログラム（p.415）を参照してください．

17 Self-Feedback/Connection／FM音源内部レジスタ

I/Oアドレス **018AH**

内部アドレス **B0H B1H B2H**

解説

Self Feedbackは，各チャネルの第一スロットの変調度を決めるデータです．つまり第一スロットは自分自身の出力を変調信号としているため，その変調度をこのデータで制御します．変調度は表に示すとおりです．

Self Feedback	0	1	2	3	4	5	6	7
変調度	OFF	$\pi/16$	$\pi/8$	$\pi/4$	$\pi/2$	π	2π	4π

Connectionは，4つのスロットをどのように組み合わせるかを指定するデータです．それらの組み合わせは全部で8種類あります（p.412図2-43参照）．

サンプルプログラム（p.415）を参照してください．

18 Channel-A,B,C Tone Period／FM音源内部レジスタ

I/Oアドレス **018AH**

内部アドレス **00H 01H 02H 03H 04H 05H**

解説

SSG音源部も3チャネルの音を発生させる事ができます．この3チャネルの音の周波数を決めるのがこのレジスタ郡です．

周波数は12bitsで構成されますので，上位4bitsをCoarse Tuneとして，下位8bitsをfine Tuneとして別のレジスタに登録します．

発信周波数次の式で求められます．

fM：入力クロック周波数(3.9936MHz)

n：プリスケーラコントロールで定められる分周数(4)

Tp：12bitsで構成される値(0～2047)

f tone = fM/(8×n×Tp)

サンプル　サンプルプログラム（p.415）を参照してください．

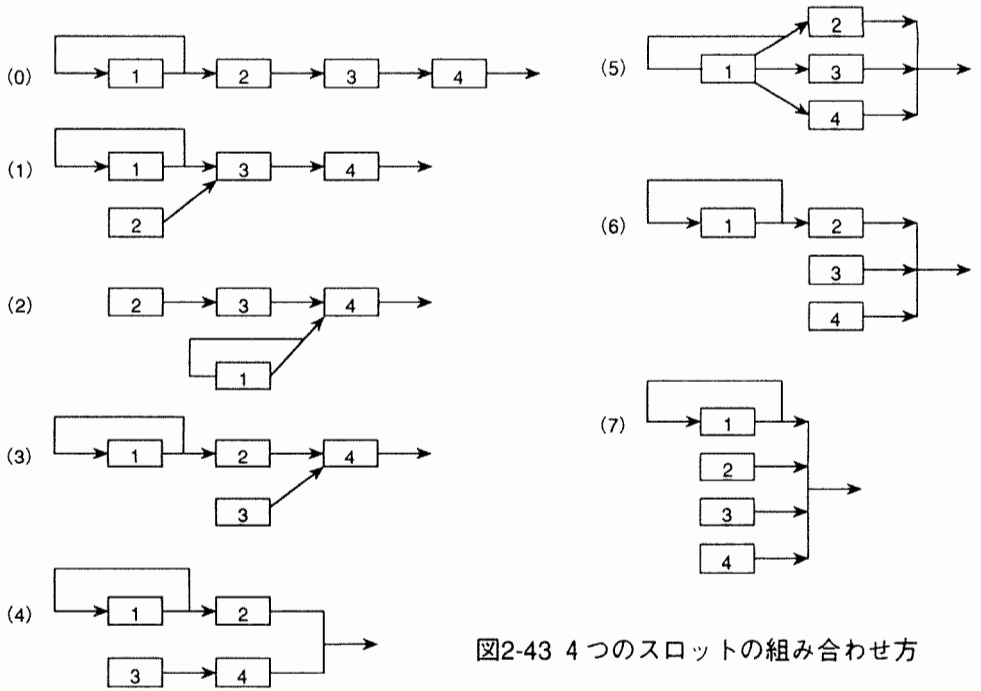



図2-43 4つのスロットの組み合わせ方

D3 C	D2 ATT	D1 ALT	D0 HLD	エンベロープ形状
0	0	X	X	
0	1	X	X	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

図2-44 エンベロープ形状

19 Noise Period　／FM音源内部レジスタ

I/Oアドレス　018AH

内部アドレス　06H

解説　ノイズジェネレータをコントロールします．ノイズ源は系列長の擬似ランダムノイズです．系列の発生をコントロールするのがこのレジスタの働きです．系列発生周期(ノイズ周波数)はレジスタの下位5bitsで決められ，次の式で求める事ができます．

fM：入力クロック周波数(3.9936MHz)
n：プリスケーラコントロールで定められる分周数(4)
Np：5bitsで構成される値(0～31)
f noise = fM/(8×n×Np)

サンプル　サンプルプログラム（p.415）を参照してください．

20 ミキサーとI/Oのコントロール　／FM音源内部レジスタ

I/Oアドレス　018AH

内部アドレス　07H

解説　このレジスタは，3つのノイズ・トーンミキサーと2つの汎用I/Oポートの入出力をコントロールします．ミキサーは，トーンとノイズの出力スイッチになっていまして，割り当てビットを1にする事で出力されます．I/Oポートの入出力制御は，bit6およびbit7に割り当てられていますが，PC-9801-26KなどのFM音源ではI/O PortAは入力に，I/O PortBは出力に割り当てられていますので，上位2bitsの並びは必ず10となります．

サンプル　サンプルプログラム（p.415）を参照してください．

21 Channel A,B,C AmplitudeとMode／FM音源内部レジスタ

I/Oアドレス　018AH

内部アドレス　08H 09H 0AH

解説　このレジスタで，A，B，Cの３つのチャネルに対するD/Aコンバータの振幅をコントロールします．第4bitが0の時は固定振幅モードとなり，残りの4bitsで決められる値をとります．第4bitが1の場合可変振幅モードとなり，後述するエンベロープコントロールの出力で決まる振幅となります．なお，D/Aコンバータは5bitsの能力を持っています．固定振幅モード時の4bitsのデータがD/Aコンバータへの入力の上位4bitsとして扱われま

す.

サンプル　サンプルプログラム（p.415）を参照してください.

22 Envelop Period ／FM音源内部レジスタ

I/Oアドレス **018AH**

内部アドレス **0BH 0CH**

解説　変化のあるエンベロープを発生させるために，二つのエンベロープコントロールがあり，そのひとつが0BH,0CHで与えられたデータを元に設定される，エンベロープ周波数です．エンベロープ周波数は次のように求められます．

fM：入力クロック周波数(3.9936MHz)
n ：プリスケーラコントロールで定められる分周数(4)
Ep：16bitsであらわされる値です．値は0CHが上位8bitsに0BHが下位8bitsにあたります．
fE = fM/(128×n×Ep)

サンプル　サンプルプログラム（p.415）を参照してください.

23 Envelop形状 ／FM音源内部レジスタ

I/Oアドレス **018AH**

内部アドレス **0DH**

解説　エンベロープコントロールのもうひとつは，エンベロープの形状を決定するこのレジスタです．エンベロープの形状は0DHの下位4bitsで決定されまして，各ビットの機能は次のように定義されています（p.412図2-44参照）．

C　　：1に設定されると，HLDのデータで定められる状態になります，0に設定されると1サイクル終了後エンベロープジェネレータは全て0にリセットされます．
ATT　：1に設定された場合はエンベロープジェネレータは全て0から全て1にカウントアップし，0に設定されたのであれば1から全て0へカウントダウンされます．
ALT　：1に設定された場合，各サイクル毎にエンベロープカウントの方向を反転させます．
HOLD：1に設定された場合は，エンベロープカウンタを1サイクル終了後カウンタの最終値をホールドします．ただし，ALTも1のときは，直前の初期カウント値にリセットされます．

サンプル　サンプルプログラム（p.415）を参照してください.

24 I/O Port Data A,B ／FM音源内部レジスタ

I/Oアドレス　018AH

内部アドレス　0EH 0FH

解説　SSG音源部に搭載されている汎用I/Oポートは，ジョイスティックインターフェイスに使用されています．0EHが入力に，0FHが出力に割り当てられています．詳しい事はJOY STICKの項をご覧ください．

サンプル　サンプルプログラム（p.415）を参照してください．

●WAIT

OPNのアドレスレジスタが20HからB2Hの部分にアクセスする場合には，ウェイトを入れないと正常に動作しません．

①FM音源ボードのI/Oポートに入出力コマンドを行うと，7クロックのウェイトがハードウェアにより自動的に入ります

②ライトアドレスでアドレスレジスタを送った後，43クロック以上のソフトウェアでのウェイトもしくは，リードステータスによりBUSYフラグが0になったことを確認してから，ライトデータでデータを送る必要があります

③ライトデータから次のライトデータまでには208クロック以上のウェイトが必要です

と，9801テクニカルマニュアルに記述されていますが，これはPC-8801SRのFM音源のウェイトを2.5倍した数値のようですので，最近の速い98シリーズでは必ずしも十分とはいえません．クロック数によるウェイトは486などのCPUでは正しいウェイトが取れないこともあるので，時間でウェイトを求めてみましょう．

まず，NECで最初にFM音源が搭載されたパソコンPC-8801SRでのFM音源のウェイトを見てみますと，次のようになっていました．

①ライトアドレスでアドレスレジスタを送った後には17クロック以上のウェイトが必要
②ライトデータから次のライトデータまでには83クロック以上のウェイトが必要

ここで，PC-8801SRのシステムクロックは4MHzでしたから，1クロックに要する時間は，

$$1/(4\times10^6)=0.25\mu sec$$

したがって，アドレスを送った後のウェイト時間は，

$17 \times 0.25 = 4.25$（μ sec）

データを送った後のウェイトは，

$83 \times 0.25 = 20.75$（μ sec）

となります．

この時間より長い時間ウェイトをとればよいということになります．I/Oポートの5FHに何かデータを出力することにより，最低0.6 μ secのウェイトが取れることが知られていますので，これを使うことにします．

実際にプログラムする場合のウェイトの入れ方は以下のようになります．

①アドレスを送る前にBUSYフラグをチェックする
②BUSYでなければ，アドレスデータを送る
③BUSYフラグをチェックする
④I/Oポート5FHに何か出力することによるウェイトを20回以上繰り返す
⑤レジスタデータを送る

これを繰り返してFM音源にデータを送っていきます．実際に音楽ドライバなどで使われているI/Oポート5FHによるウェイトは12回程度ですが，余裕をもって行うため20回としました．

■サンプルプログラム

```
/*
 * $Author: Suge $
 * $Date: 93/12/13 06:33:09 $
 * $Revision: 1.2 $
 * $Log:  fm.c $
 * Revision 1.2   93/12/13   06:33:09   Suge
 *  F-Numberの指定が間違っていたのを修正
 *
 * Revision 1.1   93/10/31   00:14:18   Suge
 * Initial revision
 *
 */

#include <dos.h>
void send_fm( unsigned char address_reg, unsigned char data_reg );

int main( void )
{
     unsigned char fmd00[]={ 15,82,50,2,65,27,0,0,94,94,94,158,15,7,8,7,5,5,5,5,
                                           214,214,216,216,0,0,0,0,60 };
      unsigned char i;

      /* FM音源部のサンプル */
      for ( i = 0; i < 28; i ++ )
          send_fm( 0x30 + i * 4, fmd00[ i ] );
```

```
	send_fm( 0xb0, fmd00[ i ] );
	send_fm( 0xa4, 0x1a );
	send_fm( 0xa0, 0x6a );
	send_fm( 0x28, 0xf0 );
	/* 3秒間待つ */
	delay( 3000 );
	/* 音を止める */
	send_fm( 0x28, 0 );

	/* SSG音源部のサンプル */
	/*トーン発信周波数を440Hzに設定 */
	send_fm( 0x01, 0x01 );
	send_fm( 0x00, 0x1c );

	/*ノイズ発信周波数を5000Hzに設定 */
	send_fm( 0x01, 0x01 );
	send_fm( 0x00, 0x1c );

	/*エンベロープ周波数を2Hzに設定*/
	send_fm( 0x0c, 0x0f );
	send_fm( 0x0b, 0x3c );

	/*エンベロープ形状を三角波に設定*/
	send_fm( 0x0d, 0x0e );

	/*エンベロープを有効にする*/
	send_fm( 0x08, 0x10);

	/*トーンとノイズを出力*/
	send_fm( 0x07, 0xb6 );
	/* 5秒間待つ */
	delay( 5000 );

	/*トーンを出力*/
	send_fm( 0x07, 0xbe );
	/* 5秒間待つ */
	delay( 5000 );

	/*ノイズを出力*/
	send_fm( 0x07, 0xb7 );
	/* 5秒間待つ */
	delay( 5000 );

	/* 音を止める */
	send_fm( 0x07, 0xbf );

	return 0;
}
void send_fm( unsigned char address_reg, unsigned char data_reg )
{
	const unsigned int WRITE_ADDRESS = 0x188;
	const unsigned int WRITE_DATA = 0x18a;
	const unsigned int READ_STATUS = 0x188;
	const unsigned int READ_DATA = 0x18a;
```

```
    unsigned char result;

    /* busy flag check */
    do {
        result = inportb( READ_STATUS );
    }while( result & 0x80 );

    /* OPNへアドレス出力 */
    outportb( WRITE_ADDRESS, address_reg );

    /* busy flag check */
    do {
        result = inportb( READ_STATUS );
    }while( result & 0x80 );

    /* wait */
    outportb( 0x5f, 0 );
    outportb( 0x5f, 0 );
    outportb( 0x5f, 0 );
    outportb( 0x5f, 0 );
    outportb( 0x5f, 0 );

    outportb( 0x5f, 0 );
    outportb( 0x5f, 0 );
    outportb( 0x5f, 0 );
    outportb( 0x5f, 0 );
    outportb( 0x5f, 0 );

    outportb( 0x5f, 0 );
    outportb( 0x5f, 0 );
    outportb( 0x5f, 0 );
    outportb( 0x5f, 0 );
    outportb( 0x5f, 0 );

    outportb( 0x5f, 0 );
    outportb( 0x5f, 0 );
    outportb( 0x5f, 0 );
    outportb( 0x5f, 0 );
    outportb( 0x5f, 0 );

    /* OPNへデータ出力 */
    outportb( WRITE_DATA, data_reg );
}
```

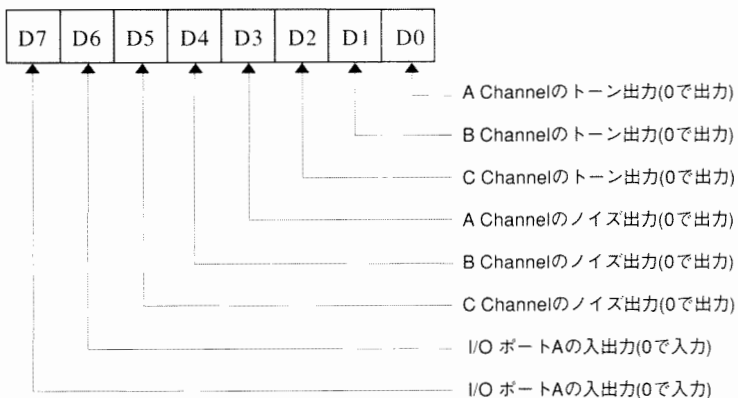
PC-9801-26/Kには，ジョイスティックインターフェースも付いています，これを使う方法を説明します．

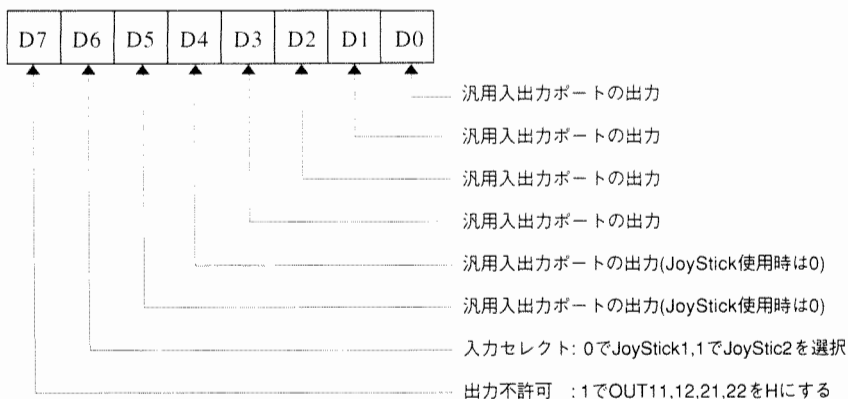
●ハードウェア解説

ジョイスティックインターフェースはOPNのI/Oポートを使用します．ビットの割り当ては次の通りです．

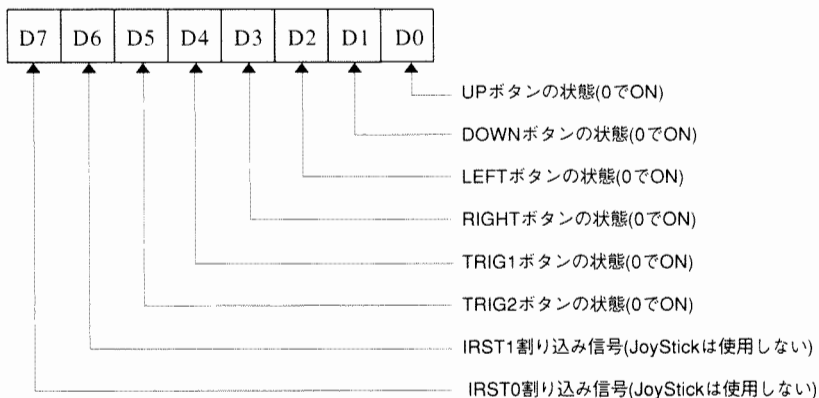
I/Oポート	内部アドレス	データ							
		D7	D6	D5	D4	D3	D2	D1	D0
A入力	0EH	IRST0	IRST1	TRIG1	TRIG2	RIGHT	LEFT	DOWN	UP
B出力	0FH	OUTE	INSL	OUT23	OUT13	OUT22	OUT21	OUT12	OUT11

ジョイスティックを使用する手順は以下の通りです．

まず，下準備としてOPNの内部アドレス07hでジョイスティックインターフェースの入出力をコントロールしていますので，そこを設定します．OPN Address 07hのビット割り当ては，

となってますので，サウンドに支障を起こさないようにまずOPN Address07hの内容を読み込んで，D7を1にD6を0にセットします．

ついで，OPN Address 0fhの設定を行います．OPN Address 0fhのビット割り当ては，

ジョイスティック1から入力する場合は8fh，ジョイスティック2の場合は0cfhをOPN Adress 0fhに出力することで使用するジョイスティックの選択と初期化が行えます．

これでジョイスティックから入力する準備ができました．

ジョイスティックからの入力はOPN Address 0ehに割り当てられています．OPN Address 0ehのビット割り当ては，

下位6bitをプログラム中で判断すればどのボタンが押されているかわかります．

■サンプルプログラム

最後にサンプルプログラムを添付しますので参考にしてください.

```
/*
 * $Author: Suge $
 * $Date: 93/12/15 11:17:24 $
 * $Revision: 1.1 $
 * $Log:   joytest.c $
 * Revision 1.1   93/12/15   11:17:24   Suge
 * Initial revision
 *
 */

#include <dos.h>

#define FM_PORT1 0x0188
#define FM_PORT2 0x018a

#define OPN_IO_A 0x0e
#define OPN_IO_B 0x0f

#define JOY_UP 0x01
#define JOY_DOWN 0x02
#define JOY_LEFT 0x04
#define JOY_RIGHT 0x08
#define JOY_TRIG1 0x10
#define JOY_TRIG2 0x20

void InitJoystick( int stick );
unsigned char GetJoystick( void );

/* Joystickのデータの変換 */
unsigned char triger( unsigned char joy_status );
unsigned char joy_key( unsigned char joy_status  );

void main( void )
{
      unsigned result;

      InitJoystick( 1 ); /* ジョイスティック 0を選択 */
      do {
            result = GetJoystick();
      /* Triger1とleftがONならば終了 */
      } while ( result != 0x14 );
}

void InitJoystick( int stick )
{
      unsigned char result;

      /* OPN I/O の設定(A:入力 B:出力 ) */
      outportb( FM_PORT1, 0x07 );
      result = inportb( FM_PORT2 );
      result &= 0x3f;         /* 上位2bitをクリアー */
      result |= 0x80;         /* 上位2bitを"10"に設定 */
```

```
    outportb( FM_PORT1, 0x07 );
    outportb( FM_PORT2, result );

    /* Connector Select ( 1: CONNECTOR1, 2:CONNECTOR2, OTHER:CONNECTOR1 ) */
    outportb( FM_PORT1, OPN_IO_B );
    if ( stick == 2 )
    {
        outportb( FM_PORT2, 0xcf );
    } else {
        outportb( FM_PORT2, 0x8f );
    }
}
unsigned char GetJoystick( void )
{
    unsigned char result;

    outportb( FM_PORT1, OPN_IO_A );
    result = inportb( FM_PORT2 );
    result ^= 0xff;         /* ビット反転 */
    result &= 0x3f;         /* 割り込み部分をマスク */
    return ( result );
}
```

■2-13-3 タイマー

OPNにはタイマーが搭載されています．このタイマーを使う方法を説明します．

●プリスケーラ

OPN Address 2dh,2eh,2fhがプリスケーラに割り当てられています．FM,SSG各音源の周波数をコントロールしているのがプリスケーラです，この3個のアドレスに関してはデータビットはなく，アドレスを指定するだけで分周数を決めます．分周数とアドレスの関係は表のようになります．

表2-44　入力クロックと内部クロックの関係

2D	2E	2F	FM音源の分周数	SSG音源の分周数	OPNに入力できる最大周波数
—	—	ON	2	1	1.4MHz
ON	—	—	6	4	4.2MHz
ON	ON	—	3	2	2.1MHz

しかし，リセット動作を保証するためにマスタクロックの分周数を6以外に設定していけないことになっていますで，以下の説明は分周数を6に指定しているものとして進めて行きます．

また，リセット時は6分周に設定されています．

●二種類のタイマー

OPNには二種類のタイマーが搭載されています．ひとつは10bitプリセッタブルタイマー(TimerA)，もうひとつは8bitプリセッタブルタイマー(TimerB)です．各タイマーは指導・停止およびフラグの制御が可能です．

◆TimerA　OPN Address 24h, 25h

10bitをプリセット値としてカウンタを動かします．そしてカウンタがオーバーフローを起こしたときにTimerAのフラグを立て，同時にプリセット値をロードします．

TimerAは，タイマー機能以外にCSMのコントロールとしても働きます．この場合は，オーバーフローが生じたときのみチャネル3の各スロットをONにして，チャネル3の全スロットの音を出力します．これにより，複合正弦波合成による音声合成が可能になります．

TimerAのオーバーフローする時間は以下のようにして求めます．

NA: 24h, 25hに設定する10bitの値

f: マスタクロック周波数(3.9936MHz)/分周数

$T = 12 \times (1024 - NA)/f$

$T = 12 \times (1024\text{-}NA)/665.6$(デフォルト設定)

◆TimerB　OPN Address 26h

TimerBは8bitをプリセット値としてカウンタを動かします．時間は以下のようにして求めます．

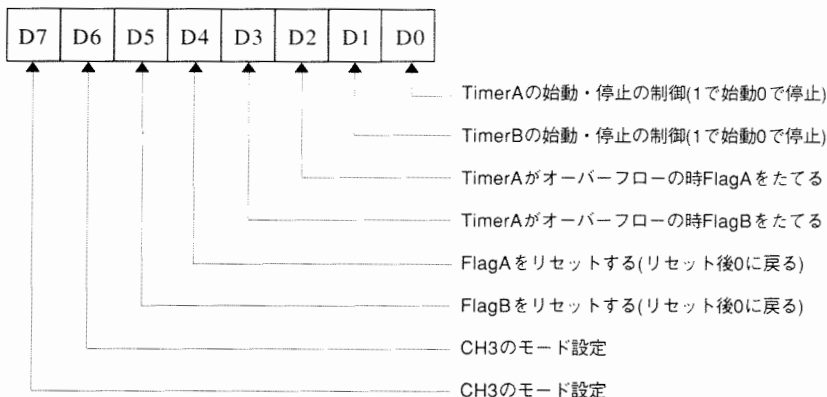
NB:26hに設定する8bitの値

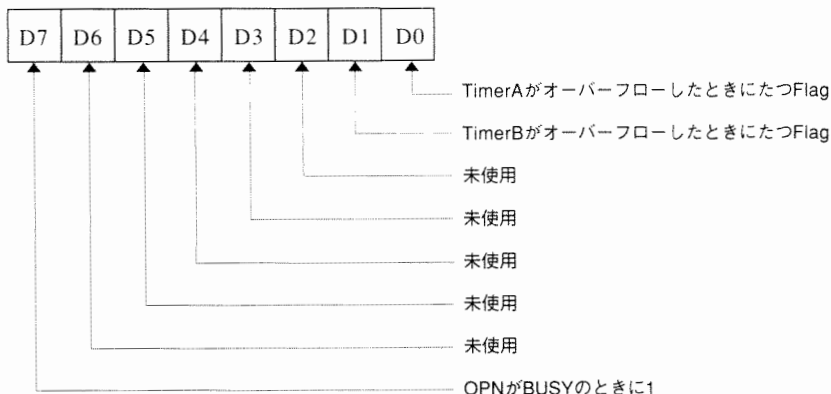
f: マスタクロック周波数(3.9936MHz)/分周数

$T = 192 \times (1024 - NA)/f$

$T = 192 \times (1024\text{-}NA)/665.6$(デフォルト設定)

●タイマーの起動

タイマーの設定は，OPN Address27Hで行います．ビット割り当ては次のようになってます．

ここでいうFlagとは，I/O Port 0188hを読んだ(Read Status)ときの内容にあらわされるものです．

Read Status時のビット割り当ては次のようになってます.

●割り込み

タイマーを利用して，CPUに割り込みをかけることができますので，これを使う方法を説明します．

◆割り込みベクタ

OPNの割り込み信号は，反転しIR131(INT6)またはIR121(INT5)に接続されています．割り込み信号は，ジャンパピンを差し替えることによって設定できます，ソフトウェアからはOPN Address 0ehの最上位bitをみることによってどちらの信号を使用しているかわかります．最上位ビットが1のときはIR131(INT6),0のときはIR121(INT5)を使用していることをあらわします．

通常はINT5を使用します(テクニカルマニュアルにはINT6を通常使用とありますが，INT6はマウスが使用していますし，FM音源を内蔵している機種でもINT5が使用されています)．ハードウェア割り込みの設定に関しては，本書の割り込みのところを参考していただいてここでは説明しないことにします．

§ 2-14 そのほかの機能

98には，これまでの項で述べてきた機能のほかにも，まだ若干の細かい機能がいくつかあります．ここでは，そのような細かい機能のことについて述べてみたいと思います．

■2-14-1 CPUリセット機能

80286以上のCPUを搭載している98には，I／Oポートを制御することでCPUのみをハードウェア的にリセットする機能があります．そのCPUリセットに関係するI／Oポートを表2-45に示しておきます．

なぜ，このようなCPUリセット機能が設けられたかというと，80286にプロテクトモードからリアルモードに移行する方法がなく，プロテクトモードからリアルモードに移るためにはCPUをリセットするしかなかったからです．そのため，CPUをリセットしてもプログラムの実行を継続することができるように，この機能が設けられたわけです．

実際にこのCPUリセット機能を使うには，まず，システムポートのSHUT0およびSHUT1ビットを操作してリセット後の動作を設定します．SHUT0＝1とすると，SHUT1の値によって，通常のリセットと同じ動作をしたり，「SYSTEM SHUTDOWN」と表示されて停止したりします．一方，SHUT0＝0とすると，リセット後次のような動作をします．

SS←[0000H：0406H]
SP←[0000H：0404H]
RETF（FAR　RET命令）

つまり，この機能を用いてリセット後処理を継続したいときには，リセット後の処理ルーチンのコードセグメント（CS）とインストラクションポインタ（IP）をスタックにプッシュし，スタックセグメント（SS）とスタックポインタ（SP）をそれぞれ[0000H：0406H]と[0000H：0404H]に格納してからリセットポートに00Hを出力すればよいわけです．

なお，80286以外のCPUでは，プロテクトモードからリアルモードへ移る方法が用意されているので，このCPUリセットは再ブートをするくらいの利用価値しかありません．

表2-45 CPUリセット関係のI／Oポート

リード／ライト	I／O アドレス	機　　能	D7	D6	D5	D4	D3	D2	D1	D0
リード	35H	ポートCの読み出し（診断用）	SHUT0	*	SHUT1	*	*	*	*	*
ライト	35H	ポートCの書き込み（一括書き込み）	SHUT0	*	SHUT1	*	*	*	*	*
	F0H	CPUリセット	0	0	0	0	0	0	0	0

＊ 設定時，値を変えないようにする

SHUT0	SHUT1	ソフトウェアリセット後の動作
1	1	通常のリセットと同じ動作をする．
1	0	「SYSTEM SHUTDOWN」と表示して停止する．
0	×	CPUリセット後，プログラムの実行を継続する

■2-14-2 アドレスバスA20ビットマスク解除

8086では，アドレスバスはA0～A19の20ビットでしたが，80286以上のCPUはA20以上のアドレスバスが存在します．が，98では，80286以上のCPUでもリセット後そのままの状態ではA20はマスクされていて効力を持ちません．表2-46に示したポートは，そのアドレスバスA20ビットのマスクを解除して有効にするためのものです．このポートに値を出力してはじめて，1Mバイト以上のメモリ空間にアクセスすることが可能になります．

表2-46 アドレスバスA20マスク解除関係のI／Oポート

リード／ライト	I／O アドレス	機　　能	D7	D6	D5	D4	D3	D2	D1	D0
ライト	F2H	A20マスク解除	0	0	0	0	0	0	0	0

第３部
H98・MATE編

PROGRAMMERS BIBLE

§3-1 H98・MATEの性能

●H98

H98（Hyper 98）は，ノーマル98とは1線を画した性能を持つ98です．その主な特長は，次のようなものです．

・ノーマル・ハイレゾ両モードに対応
・NESA（New Extended Standard Architecture）バスによる高速転送
・AGDC（Advanced GDC），E²GC（Enhanced Expanded Graphic Charger）による高速描画機能
・グラフィック256色表示に対応
・テキスト画面に16色表示可能

これらのほかに，ディスプレイの水平同期周波数24／31／50KHzに対応，I／Oへの出力値はほとんど後で読み出すことができる，使えるユーザー定義文字が多い，等の特長も持っています．

●98MATE/MULTI

98MATE（MULTI）は，98のマルチメディア（映像・音声）関係の性能を大幅に向上させたもので，主な特長としては次のようなものがあります．

・640×480ドット，256色のグラフィック表示が可能
・FM6音，SSG3音，リズム6音，ステレオPCMを持つサウンド機能

これらのほかに，H98と同様にI／Oへの出力値はほとんど後で読み出すことができるという特長もあります．

このように，H98とMATEはいずれも従来の98とは1線を画する性能を持っており，しかも互いに似ている部分もあります．そこで，この章では，主にH98とMATEの画面表示関係の拡張機能を，両者を対比させつつ解説していこうと思います．

§ 3-2 グラフィック256色表示

H98とMATEはいずれも1677万色中256色のグラフィック表示をすることができます．が，両者の256色表示時のグラフィック画面の扱い方は同じではなく，いろいろな違いがあります．そこで，以下，256色表示におけるH98とMATEの同じ点，異なる点について順次説明していきたいと思います．

■3-2-1 256色表示でのH98・MATE共通事項

256色表示においてH98とMATEに共通するものとしては，まず，標準グラフィックモード（従来の98のグラフィックと互換性のあるモード）から拡張グラフィックモード（1677万色中256色が使えるモード）へと移るモード変更のしかたがあります．拡張グラフィックモードに移るには，両者ともI／Oアドレス6AHのモードフリップフロップ2を制御します（表3-2参照）．

具体的には，C言語で拡張グラフィックモードに移るには，

```
outportb(0x6a, 0x07);            /* モード変更可 */
outportb(0x6a, 0x21);            /* 拡張モード */
```

とします．また，同じくC言語で標準グラフィックモードに移るには

```
outportb(0x6a, 0x07);            /* モード変更可 */
outportb(0x6a, 0x20);            /* 標準モード */
```

とします．

このモードフリップフロップ2は，ノーマル98では設定した値を後から読み出すことはできませんでしたが，H98およびMATEでは設定値を読み出すことができます．具体的には，まず，I／Oアドレス09A0Hに調べたいモードの番号を出力し，その後同じ09A0Hから値を読み込みます．すると，モードF／F2の設定内容によって，その入力値のFFビット（ビット0）に0または1が返されます（表3-3参照）．

たとえば，C言語で，現在のモードが標準グラフィックモードか拡張グラフィックモードか調べたいときには，

```
int ff;
outportb(0x9a0, 0x0a);              /* グラフィックモード読み出し指定 */
ff = inportb(0x9a0) & 0x01;
```

とすれば，標準グラフィックモードならff＝0，拡張グラフィックモードならff＝1になります．

H98とMATEは，拡張グラフィックモードでのパレットの設定方法も共通であり，標準グラフィックモードでのパレットと同じI／Oポートを使います．そして，設定したパレットを読み出すことも可能です．

拡張グラフィックモードでパレットを設定するには，標準グラフィックモードのときと同じように，I／OアドレスA8Hに色を設定するパレット番号を，I／OアドレスAAH，ACH，AEHにそれぞれ緑，赤，青の輝度を設定します．ただ，標準グラフィックモードのときと異なり，各ポートには0～255までの数値を指定できるので，パレット番号は256種類，色は256^3＝16,777,216種類指定できることになります．たとえば，パレット番号184に黄緑色を設定するには，緑輝度＝255，赤輝度＝128，青輝度＝0を設定すればよいので，

```
outportb(0xa8, 184);          /* パレット番号 */
outportb(0xaa, 255);          /* 緑輝度 */
outportb(0xac, 128);          /* 赤輝度 */
outportb(0xae,   0);          /* 青輝度 */
```

としてやります．ただし，H98の場合には，高速パレット書き込みモードになっていないとタイミングを考慮しながら設定を行わなければならないので，事前に，

```
outportb(0x6a, 0x07);         /* モード変更可 */
outportb(0x6a, 0x2b);         /* 高速書き込み */
```

として高速パレット書き込みモードにしておくようにします．

また，I／Oポートからパレットを読み出したいときには，まず，I／OアドレスA8Hに色を読み出したいパレット番号を設定し，I／OアドレスAAH，ACH，AEHからこの順番で読み込んでいきます．そうすると，それぞれ現在設定されている緑輝度，赤輝度，青輝度を読み出すことができます．ただし，H98の場合には，なぜか1回の読み出しでは正常に読み出せないことがあるので，1回空読みしてから読み出しを行うようにします．たとえば，現在パレット番号8に設定されている色を読み出すには，

```
int i, gbr, rbr, bbr;
for (i = 1; i <= 2; i++) {
    outportb(0xa8, 8);            /* パレット番号 */
    gbr = inportb(0xaa);          /* 緑輝度 */
    rbr = inportb(0xac);          /* 赤輝度 */
    bbr = inportb(0xae);          /* 青輝度 */
}
```

とすれば，gbrに緑輝度が，rbrに赤輝度が，bbrに青輝度が得られます．

さらに，H98とMATEで共通に拡張されているそのほかのI／Oポートとしては，CRTの垂直同期周波数を指定するI／Oポートがあります．このポートは，CRTの垂直同期周波数を，通常の640×400ドット表示のための24KHzにするか，あるいは640×480ドット表示のための31KHzにするかを指定するものです．具体的には，垂直同期周波数を24KHzにするにはI／Oアドレス09A8Hに00Hを，31KHzにするには01Hを出力します．

C言語で実際にこの設定をするには，水平同期周波数を24KHzにするには，

```
outportb(0x09a8, 0);            /* 24KHz */
```

とし，31KHzにするには

```
outportb(0x09a8, 1);            /* 31KHz */
```

とします．

表3-1　H98・MATE共通の拡張I／Oポート

リード／ライト	I／O アドレス	機能	D7	D6	D5	D4	D3	D2	D1	D0
リード	A4H	表示画面の読み出し	×	×	×	×	×	×	×	DP
	A6H	描画画面の読み出し	×	×	×	×	×	×	×	WP
	AAH	緑輝度の読み出し	←緑輝度→							
	ACH	赤輝度の読み出し	←赤輝度→							
	AEH	青輝度の読み出し	←青輝度→							
	09A0H	モードF／F2の読み出し	×	×	×	×	×	×	×	FF
	09A8H	水平同期周波数の読み出し	×	×	×	×	×	×	×	HF
ライト	A4H	表示画面の指定＊	0	0	0	0	0	0	0	DP
	A6H	描画画面の指定	0	0	0	0	0	0	0	WP
	A8H	パレット番号の書き込み	←パレット番号→							
	AAH	緑輝度の書き込み	←緑輝度→							
	ACH	赤輝度の書き込み	←赤輝度→							
	AEH	青輝度の書き込み	←青輝度→							
	6AH	モードフリップフロップ2のコントロール（表3-2）	ADR6	ADR5	ADR4	ADR3	ADR2	ADR1	ADR0	DT
	09A0H	読み出すモードF／F2指定（表3-3）	←F/F2番号→							
	09A8H	水平同期周波数の指定	0	0	0	0	0	0	0	HF

＊モードF／F2によって表裏ページが連続になっているときは無効

表3-2　H98，MATEのモードF／F2（I／Oアドレス6AH）

出力する値	意　　味	解　　説
００Ｈ	８色グラフィックモード	標準グラフィックモードでの表示モードの選択
０１Ｈ	16色グラフィックモード	
０４Ｈ	ＧＲＣＧ互換モード*	ＥＧＣの動作モードの選択
０５Ｈ	ＥＧＣ拡張モード*	
０６Ｈ	拡張モード変更不可	拡張グラフィックモード変更の可否の選択
０７Ｈ	拡張モード変更可	
２０Ｈ	標準グラフィックモード*	256色拡張グラフィックモード使用の有無
２１Ｈ	拡張グラフィックモード*	
２６Ｈ	通常表示**	全画面の階調を反転させるか否かの選択
２７Ｈ	全画面反転表示**	
２８Ｈ	通常重ね合わせ**	テキスト・グラフィック画面の重ね合わせ形式の選択
２９Ｈ	演算付き重ね合わせ**	
２ＡＨ	通常パレット書き込み**	拡張モードでのパレット書き込み方法の選択
２ＢＨ	高速パレット書き込み**	
２ＣＨ	オーバースキャンカラーなし**	表示区間の周囲の非表示区間の色指定の有無の指定
２ＤＨ	オーバースキャンカラーあり**	
４０Ｈ	ＣＲＴモード	テキスト画面の1ドットの横ずれの制御．ＣＲＴモードのときずれる
４１Ｈ	プラズマディスプレイモード	
６２Ｈ	プレーン形式**	ＶＲＡＭ形式の選択
６３Ｈ	パックトピクセル形式**	
６８Ｈ	表ページ・裏ページは別個*	表裏ページの表示形式の選択．別個のとき１ページ分を超えた部分はラップラウンドする
６９Ｈ	表ページ・裏ページは連続*	
６ＣＨ	上位ビットが左方向**	ＶＲＡＭのビットマップ形式の選択
６ＤＨ	上位ビットが右方向**	
８４Ｈ	ＧＤＣ　2.5MHzモード	グラフィックＧＤＣの動作周波数の選択．５ＭＨｚモードにするには８３Ｈと８５Ｈを両方出力する．周波数を変更したらＳＹＮＣコマンドの再設定が必要
８３Ｈと８５Ｈ	ＧＤＣ　5MHzモード	

*　拡張モード変更可のときのみ有効
** H98で拡張モード変更可のときのみ有効

表3-3　モードF／F2の読み出し方法（I／Oアドレス09A0H）

0 9 A 0 Hに出力する値	09A0Hからの入力値のbit0	現在の設定状態
0 4 H	0	8色グラフィックモード
	1	16色グラフィックモード
0 5 H	0	CRTモード
	1	プラズマディスプレイモード
0 7 H	0	GRCG互換モード
	1	EGC拡張モード
0 8 H	0	拡張モード変更不可
	1	拡張モード変更可
0 A H	0	標準グラフィックモード
	1	拡張グラフィックモード
0 B H	0	プレーン形式*
	1	パックトピクセル形式*
0 D H	0	表ページ・裏ページは別個
	1	表ページ・裏ページは連続
1 1 H	0	通常表示*
	1	全画面反転表示*
1 2 H	0	オーバースキャンカラーなし*
	1	オーバースキャンカラーあり*
1 3 H	0	通常重ね合わせ*
	1	演算付き重ね合わせ*
1 7 H	0	上位ビットが左方向*
	1	上位ビットが右方向*
1 8 H	0	通常パレット書き込み*
	1	高速パレット書き込み*

＊H98のみ

■3-2-2 ――― H98の256色表示

H98で256色表示をするときは基本的に，VRAM形式をプレーンモードにして，グラフィックチャージャを使って描画を行います．その際，動作モードがGRCG互換モードでは4プレーンまでしかアクセスできませんから，EGC拡張モードにします．

そして，標準グラフィックモードのときと同じように1プレーンずつCPUから読み書きを行いたいときは，アクセスするプレーン番号が変数plnに指定されているとして，

```
outportb(0x7c, 0x80);                       /* EGC ON */
outportb(0x6a, 0x07);                       /* モード変更可 */
outportb(0x6a, 0x05);                       /* EGC拡張モード */
outport(0x04a4, 0x0000);                    /* CPUデータ */
outport(0x04a8, 0xffff);                    /* マスクなし */
outport(0x04ac, 0x0000);                    /* シフトなし */
outport(0x04ae, 0x000f);                    /* ビット長=16 */

outport(0x04a0, ~(0x0001 << pln));          /* 書き込みプレーン指定 */
outport(0x04a2, 0x00ff | pln * 0x100 );     /* 読み込みプレーン指定 */
```

とすると，指定したプレーン番号のプレーンがセグメントアドレスA800Hに現れ，標準グラフィックモードでのプレーンと同じように扱うことができます（4-1-2. EGCの項参照）．ただし，標準グラフィックモードのときのように複数のプレーンを異なるメモリアドレスに同時に存在させることはできません．

また，8プレーン同時アクセスを行いたいときは，たとえばRMWモード相当のROPをフォアグラウンドカラーを指定して行うときは，

```
outportb(0x7c, 0x80);                       /* EGC ON */
outportb(0x6a, 0x07);                       /* モード変更可 */
outportb(0x6a, 0x05);                       /* EGC拡張モード */
outport(0x04a0, 0xff00);                    /* 8プレーン同時 */
outport(0x04a4, 0x0cac);                    /* RMWモード相当 */
outport(0x04a8, 0xffff);                    /* マスクなし */
outport(0x04ac, 0x0000);                    /* シフトなし */
outport(0x04ae, 0x000f);                    /* ビット長=16 */

outport(0x04a2, 0x40ff | pln * 0x100 );     /* 読み込みプレーン指定 */
```

とします．

■サンプルプログラム

256色モードの図形を描画します．

```
#include <stdio.h>
#include <dos.h>
#include <math.h>
#include <conio.h>

void pset( int, int, int);
```

```
void main( void )
{
    int i, x, y, cl, gpar;

    clrscr();
    outportb(0x6a, 0x07);           /* モード変更可 */
    outportb(0x6a, 0x21);           /* 拡張グラフィックモード */
    outportb(0x6a, 0x62);           /* プレーン */
    outportb(0x6a, 0x2b);           /* パレット高速書き込み */
    outportb(0x6a, 0x06);           /* モード変更不可 */
    outportb(0xa2, 0x4b);           /* GDC CSRFORMコマンド */
    outportb(0xa0, 0);              /* L/R = 1 */
    outportb(0x68, 8);              /* モードF/F1 */
    outportb(0xa2,0xd);             /* 表示開始 */

    for (i = 0; i < 256; i++) {
        outportb(0xa8, i);          /* 256色モードのパレット設定 */
        outportb(0xaa, i);
        outportb(0xac, i);
        outportb(0xae, 0);
    }

    outportb(0x6a,7);               /* EGCモード変更可 */
    outportb(0x6a,5);               /* EGC拡張モード */
    outportb(0x7c,0x80);            /* EGC ON */
    outport(0x04a0,0xff00);         /* 8プレーン同時 */
    outport(0x04a2,0x40ff);         /* パターンデータ = フォアカラー */
    outport(0x04a4,0x0cac);         /* ROP, RMW相当 */
    outport(0x04a8,0xffff);         /* マスクなし */
    outport(0x04ac,0x0000);         /* シフトなし */
    outport(0x04ae,0x000f);         /* ビット長=16 */
    for (y = 0; y < 400; y++) {     /* 図形描画 */
        for (x = 0; x < 640; x++) {
            gpar = x + y + 48.0 * sin(x / 30.0);
            cl = gpar % 256;
            if (((gpar / 256) % 2) == 1) cl = 255 - cl;
            pset(x, y, cl);
        }
    }
    outportb(0x7c,0);               /* EGC OFF */
    outportb(0x6a,4);               /* GRCG互換モード */
    outportb(0x6a,6);               /* EGCモード変更不可 */
}

void pset(int x, int y, int cl)     /* 点描画関数 */
{
    int vadd, vdta;

    vadd = y * 80 + x / 8;
    vdta = 0x80 >> (x % 8);
    outport(0x04a6, cl);            /* フォアカラーセット */
    pokeb(0xa800, vadd, vdta);
}
```

■3-2-3 MATEの256色表示

MATEでの256色表示をする拡張グラフィックモードでは，標準グラフィックモードやH98の拡張グラフィックモードとはかなり異なるVRAM構成となります．プレーン0とプレーン1相当の領域（セグメントアドレスA800HおよびB000H）は，VRAMの指定された一部分が割り当てられるVRAMウィンドウ（のぞき窓）になります．プレーン2相当の領域（同B800H）は使われなくなり，プレーン3相当の領域（同E000H）は，VRAM上のVRAMウィンドウの位置を指定したり，EGCを制御したりするためのメモリマップトI／O（メモリアクセスによってI／Oアクセスを行うメモリ領域）になります．その各種制御を行うためのメモリマップトI／Oのうち，機能が判明しているものを表3-4に示しておきます．

表3-4 MATEのメモリマップトI／O

メモリアドレス	機　　能	データ幅	値の範囲・意味
Ｅ０００４Ｈ	ＶＲＡＭウィンドウ#0バンク位置指定	２バイト	００００Ｈ～０００ＦＨ
Ｅ０００６Ｈ	ＶＲＡＭウィンドウ#1バンク位置指定	２バイト	００００Ｈ～０００ＦＨ
Ｅ０１００Ｈ	ＶＲＡＭへの書き込み様式の指定	１バイト	００Ｈ：パックトピクセル ０１Ｈ：プレーン
Ｅ０１０２Ｈ	F00000HにＶＲＡＭ全体を出現させるか否かの指定	２バイト	００００Ｈ：出現させない ０００１Ｈ：出現させる

(参考文献　こうのたけし・小高照真：PC-9821の拡張機能解析,ざべ1993年9月号)

表中の用語のうち，VRAMウィンドウ＃0というのはセグメントアドレスA800Hからの32KB，VRAMウィンドウ＃1というのは同じくB000Hからの32KBの領域です．実際のVRAMへのアクセス方法は，まずVRAMウィンドウ＃0あるいは＃1のVRAM上の位置を指定してやります．具体的にはメモリマップトI／Oのオフセットアドレス0004Hあるいは0006Hにバンク番号を指定しますが，ここに指定するバンク番号は32KB（32768バイト）単位で，VRAM全体のサイズは512KBなので，バンク番号としては0000H～000FHを指定することになります．

VRAMウィンドウの位置を指定したら，VRAMウィンドウに対する読み書きを行えばVRAMの指定した位置に対する読み書きができますが，VRAMにどのようなデータを書きこんだらよいかはVRAM構成がパックトピクセルかプレーンかによって異なります．MATEの拡張グラフィックモードでの標準のVRAM構成はパックトピクセルなので，以下，パックトピクセルの場合のVRAMの扱い方について述べていきます．

VRAMがパックトピクセル形式の場合には，VRAM上の1バイトが画面上の1ドットに対応し，その1バイトがドットのパレット番号を表します．1バイト（＝8ビット）でパレット番号を指定するのですから，指定できるパレット番号は0から255の256種類となります．また，1バイトが1ドットに対応するので，98の横のドット数は640ドットですから，1ライン当たりのVRAM容量は640バイトになります．したがって，拡張グラフィックモードで（x，y）の位置に点を描画したいときには，

ADR＝（VRAMの先頭アドレス）＋y×640＋x

で計算されるアドレスに，描画したい点のパレット番号を書き込めばよいことになります．具体的な書き込み方としては，次のようにします．

まず，上のようにして計算したADR（VRAM上のアドレス）を32768で割った値をメモリマップトI／OのVRAMウィンドウ#0バンク位置指定の部分に書き込んでやります．そうすると，今書き込みたい部分を含んだVRAMの一部分（32KB）がVRAMウィンドウ#0（セグメントアドレスA800H）に現れます．それから，セグメントアドレスA800H，オフセットアドレスは上のADRの32768に対する剰余（割ったときのあまり，C言語ならadr　%　32768）の部分に描画したい点のパレット番号を書き込んでやります（サンプルプログラム参照）．

もちろん，VRAMウィンドウ#0の代わりにVRAMウィンドウ#1を使っても構いません．その場合，指定したVRAM部分が現れるのはセグメントアドレスB000Hとなります．

■サンプルプログラム

256色モードの図形を描画します．

```
#include <stdio.h>
#include <dos.h>
#include <math.h>
#include <conio.h>

void pset(int, int, int);
void setpal(int, int, int, int);

void main(void)
{
    int x, y, i, cl, gpar;

    clrscr();
    outportb(0x6a,0x07);              /* 拡張モード変更可 */
    outportb(0x6a,0x21);              /* 拡張グラフィックモード */
    outportb(0xa2,0xd);               /* 画面表示開始 */
    pokeb(0xe000, 0x100, 0x00);       /* パックトピクセル */

    for (i = 0; i < 256; i++) setpal(i, i, i, 0);
    for (y = 0; y < 400; y++) {    /* 図形描画 */
        for (x = 0; x < 640; x++) {
            gpar = x + y + 48.0 * sin(x / 30.0);
            cl = gpar % 256;
            if (((gpar / 256) % 2) == 1) cl = 255 - cl;
            pset(x, y, cl);
        }
    }
}

void pset(int x, int y, int cl)     /* 点描画関数 */
{
    unsigned long adr;
    unsigned vadr, ioadr;

    adr = (long)y * 640L + (long)x;
    vadr = (unsigned)(adr & 0x7fff);
```

```
    ioadr = (unsigned)(adr >> 15);
    poke(0xe000, 4, ioadr);
    pokeb(0xa800, vadr, cl);
}

void setpal(int palnum, int gbr, int rbr, int bbr)      /* パレットセット関数 */
{
    outportb(0xa8, palnum);
    outportb(0xaa, gbr);
    outportb(0xac, rbr);
    outportb(0xae, bbr);
}
```

●MATEの拡張グラフィックBIOS

MATEには，拡張グラフィックモードを扱うために拡張されたグラフィックBIOSがいくつかあります．そのうち，現在までに機能が判明しているものについて以下に述べてみたいと思います．

■MATEの拡張BIOS一覧（INT　18H）

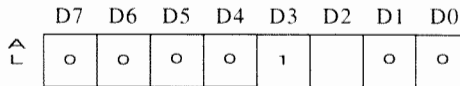
機能コード	機　　能
３０Ｈ	グラフィック画面モードの設定
３１Ｈ	グラフィック画面モードの取得
４ＤＨ	標準／拡張グラフィックモードの変更

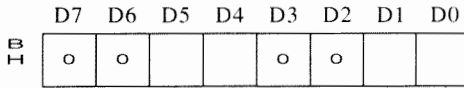
1 グラフィック画面モードの設定

割り込み　INT　18H

入　　力　AH←30H（機能コード）

AL←画面モード指定コード1

	D7	D6	D5	D4	D3	D2	D1	D0
AL	0	0	0	0	1		0	0

D2: CRTの水平スキャン周波数の指定（0：24KHz，1：31KHz）

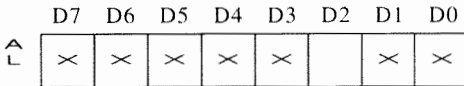
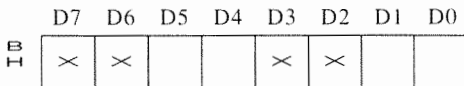
BH←画面モード指定コード2

出力 AH←結果情報（05H　：正常終了,
05H以外：不正呼び出し）

解説 GDC，モードフリップフロップ等をコントロールすることによって，グラフィック画面のモード設定を行います．BHに指定する画面モードのうち，テキスト30行表示はCRTの水平スキャン周波数が31KHzでかつ解像度モードが640×480ドットのときのみ，解像度640×480ドットモードはCRTの水平スキャン周波数が31KHzのときのみ指定可能です．グラフィックモードは，解像度モードが640×200ドットに指定されると標準グラフィックモードに，640×480ドットモードに指定されると拡張グラフィックモードに設定されます．

2 グラフィック画面モードの取得

割り込み INT　18H

入力 AH←31H（機能コード）

出力 AL←画面モード1

BH←画面モード2

解　説　グラフィック画面モードの設定で設定された現在のグラフィックモードを読み出します.

3 標準／拡張グラフィックモードの変更

割り込み　INT　18H

入　力　AH←4DH (機能コード)

CH←グラフィックモード指定コード

(00H：標準グラフィックモード,

01H：拡張グラフィックモード)

出　力　なし

解　説　標準グラフィックモード，あるいは拡張グラフィックモードを選択します．ここでの指定は，解像度モードが640×400ドットモードのときのみ有効になります．このBIOSコールは，ワークエリアの書きかえなどを除けば，次のようにするのと等価です．

C言語で，標準グラフィックモードにするには，

```
outportb(0x6a, 0x07);
outportb(0x6a, 0x20);
```

C言語で，拡張グラフィックモードにするには，

```
outportb(0x6a, 0x07);
outportb(0x6a, 0x21);
```

MATEの場合，現在のグラフィックモードはI／Oポートから直接読み出せるので，このI／O直接制御をしてもそれほど不都合はないと思われます．

§3-3 テキスト16色表示

H98は，テキスト画面に16色の色を表示することができます．この機能は，MS-DOS等でサポートされていますが，ハードウェアがサポートしている機能のごく一部を使っているにすぎません．そこでこの項では，テキスト16色モードのハードウェア構成と，ハードウェアを直接制御して拡張テキストを扱う方法について述べてみたいと思います．

ノーマル98では，テキスト画面は文字エリアが4KB×2＝8KB，アトリビュートエリアは偶数番地にしかメモリが存在しないため2KB×2＝4KBの，合計12KBのテキストVRAMによって表示を行っていました．しかし，H98では，アトリビュートエリアの奇数番地にもメモリが存在しており，文字エリア8KB＋アトリビュートエリア8KBの16KBのVRAMでの表示となります．といっても，通常の状態ではアトリビュートエリアの奇数番地に何が書いてあっても無視されてしまいます．これを有効にするには，I／Oアドレス6EHのモードフリップフロップ3を制御してやる必要があります．そこで，そのモードフリップフロップ3の構成を表3-5に示します．

H98の拡張テキストモードは，1文字当たりのアトリビュートデータの量，アトリビュートデータのビット配置，拡大表示の有無などをある程度の自由度で選択できるので，その組み合わせによって非常に多くのモードが実現できます．そこで，以下，それらのモードについて具体的に述べていきます．

●1バイト拡張テキスト

1バイト拡張テキストは，1文字当たりのアトリビュートデータは1バイトのままで，文字の色を16色中から選べるようにするなどの拡張機能を利用することができるものです．この1バイト拡張テキストには，3つの種類がありますが，そのときのアトリビュート形式はそれぞれ表3-6のようになります．

これら3つの拡張テキストはそれぞれに得意な用途があるので，たとえば，いろいろな特殊効果を付けながら文字の色を16色中から選びたいときは拡張テキスト1，背景にも色を付けたいときには拡張テキスト2，バーチカルライン，アンダーライン，ミドルラインなどを使いたいときには拡張テキスト3というように使い分けるといいでしょう．

●2バイト拡張テキスト

2バイト拡張テキストは，1文字当たりのアトリビュートデータを2バイトにすることによって，より豊富な表現を可能にするものです．この2バイト拡張テキストモードに移行するには，モードF／F3（I／Oアドレス6EH）に0FHを出力します．

2バイト拡張テキストモードには，1バイトテキストモード（標準テキスト，1バイト拡張テキスト1，2，3）にそれぞれ対応する4つのモードがあります．それらのモードは，1バイト拡張テキストのときと異なり，1文字ごとに指定することができます．具体的には，アトリビュートデータの2バイト目のビット0，1でアトリビュートモードを指定します．それら4つのアトリビュートモードでのアトリビュ

ートデータの2バイト目の形式を表3-7に示します．なお，アトリビュートデータの1バイト目は，それぞれのモードの1バイト拡張テキストのときと同じ形式になっています．

表3-5　H98のモードF／F3（I／Oアドレス6EH）

出力する値	意　　味	解　　説
0 2 H	拡張モード変更不可	以下のモードF／F3の変更の可否の選択
0 3 H	拡張モード変更可	
0 8 H	標準テキスト	1バイトアトリビュートモードでの拡張アトリビュートモードの指定（拡張モード変更可かつ1バイトアトリビュートモードのときのみ有効）
0 9 H	1バイト拡張テキスト1	
0 A H	1バイト拡張テキスト2	
0 B H	1バイト拡張テキスト3	
0 C H	拡大表示不可	アトリビュート指定による拡大表示の可否の選択（拡張モード変更可のときのみ有効）
0 D H	拡大表示可	
0 E H	1バイトアトリビュート	1文字当たりのアトリビュートのバイト数の指定（拡張モード変更可のときのみ有効）
0 F H	2バイト拡張アトリビュート	
1 4 H	文字表示あり	テキスト画面の文字表示の有無の選択（拡張モード変更可のときのみ有効）
1 5 H	文字表示なし	

表3-6　1バイトアトリビュートモードでのアトリビュート形式

テキストモード	アトリビュート形式
標準テキスト	D7 D6 D5 D4 D3 D2 D1 D0 G1 R1 B1 VLBG UL RV BL ST G1　：文字の緑 R1　：文字の赤 B1　：文字の青 VLBG：バーチカルラインの有無／簡易グラフ UL　：アンダーラインの有無（0：なし，1：あり） RV　：リバース（反転）（0：通常表示，1：反転） BL　：ブリンク（点滅）の有無（0：なし，1：あり） ST　：シークレット（0：シークレット，1：表示）
1バイト拡張テキスト1	D7 D6 D5 D4 D3 D2 D1 D0 BL SC SL UL I1 R1 G1 B1 BL　：ブリンク（点滅）の有無（0：なし，1：あり） SC　：色の指定（0:I1，R1，G1，B1は文字の色指定， 1:I1，R1，G1，B1は背景の色指定） SL　：サイドラインの有無（0：なし，1：あり） UL　：オーバーラインの有無（0：なし，1：あり） I1　：インテンシティ R1　：赤 G1　：緑 B1　：青
1バイト拡張テキスト2	D7 D6 D5 D4 D3 D2 D1 D0 BL R2 G2 B2 I1 R1 G1 B1 BL　：ブリンク（点滅）の有無（0：なし，1：あり） R2　：背景の赤 G2　：背景の緑 B2　：背景の青 I1　：文字のインテンシティ R1　：文字の赤 G1　：文字の緑 B1　：文字の青
1バイト拡張テキスト3	D7 D6 D5 D4 D3 D2 D1 D0 R1 G1 B1 SC BL ML VL UL R1　：赤 G1　：緑 B1　：青 SC　：色の指定（0:R1，G1，B1は文字の色指定， 1:R1，G1，B1は背景の色指定） BL　：ブリンク（点滅）の有無（0：なし，1：あり） ML　：ミドルラインの有無（0：なし，1：あり） VL　：バーチカルラインの有無（0：なし，1：あり） UL　：アンダーラインの有無（0：なし，1：あり）

表3-7 2バイト拡張アトリビュートモードでの2バイト目のアトリビュート形式

テキストモード	2バイト目のアトリビュート形式
2バイト拡張テキスト0 （標準テキストに対応）	D7: I2 / D6: R2 / D5: G2 / D4: B2 / D3: VL / D2: I1 / D1: 0 / D0: 0 I2　：背景のインテンシティ R2　：背景の赤 G2　：背景の緑 B2　：背景の青 VL　：バーチカルラインの有無（0：なし，1：あり） I1　：文字のインテンシティ D1＝0，D0＝0はモード0の指定
2バイト拡張テキスト1	D7: I2 / D6: R2 / D5: G2 / D4: B2 / D3: VL / D2: × / D1: 0 / D0: 1 I2：　背景のインテンシティ R2　：背景の赤 G2　：背景の緑 B2　：背景の青 VL　：バーチカルラインの有無（0：なし，1：あり） D1＝0，D0＝1はモード1の指定
2バイト拡張テキスト2	D7: × / D6: × / D5: × / D4: × / D3: VL / D2: × / D1: 1 / D0: 0 VL　：バーチカルラインの有無（0：なし，1：あり） D1＝1，D0＝0はモード2の指定
2バイト拡張テキスト3	D7: I2 / D6: R2 / D5: G2 / D4: B2 / D3: × / D2: I1 / D1: 1 / D0: 1 I2：背景のインテンシティ R2　：背景の赤 G2　：背景の緑 B2　：背景の青 I1　：文字のインテンシティ D1＝1，D0＝1はモード3の指定

第4部
資料編

PROGRAMMERS BIBLE

§ 4-1 GRCG・ECG

■4-1-1 GRCG

GRCG（GRaphic CharGer）というのは，膨大な量のデータを，重ね合わせなどの複雑な処理をしながら書きこむ必要があることが多い，グラフィック処理を高速化するためのLSIで，いわゆるグラフィックアクセラレータの一種です．その主な特徴は，次のようなものです．

1）ハードウェア的に最大4プレーンへの同時アクセスが可能．
2）ハードウェアによる重ね合わせ処理が可能．
3）ハードウェアによる特定色領域の抽出が可能．

1）についてですが，GRCGを使えば，CPUが1回データの書き込みを行うだけで最大4枚のプレーンに対して同時に書き込みが行われるので，単純に考えればデータの書き込み速度が最大4倍になり，グラフィックデータの書き込みが高速になります．

また，2）についてですが，GRCGは，ドット単位の任意の領域だけにグラフィックデータを書き込み，それ以外の領域についてはVRAMに元からあったデータを残すという重ね合わせ処理をハードウェアで高速に行う機能を持っています．そのため，ドット単位の任意の領域を単純なパターンで埋めるようなとき（たとえば塗りつぶし四角形を描くときなど）にはGRCGは非常に有効になります．

3）の特定色領域の抽出というのは，特定のパレット番号の色を別のパレット番号の色に変換するときなどに有用なものです．

GRCGは基本的に，CPUとVRAMの間に入ってデータの加工を行うものです．通常の状態ではGRCGは動作しない状態にされており，CPUがVRAMに対して書き込みを行ったデータはそのままVRAMに書き込まれ，CPUがVRAMからデータを読み込めばVRAMのデータがそのままCPUに渡されます．ところが，指定によってGRCGがONにされると，CPUとVRAMに直接のつながりはなくなり，CPUとVRAMは必ずGRCGを介してデータの受け渡しを行うようになります（図4-1参照）．つまり，GRCGがONのときにCPUがVRAMに対して書き込みを行うと，GRCGがいったんそのデータを受け取り，そのデータを加工してから最大4枚のプレーンに対して同時に書き込みを行うのです．

GRCGを制御するためのI/Oポートを表4-1に示します．

●GRCG OFFの場合

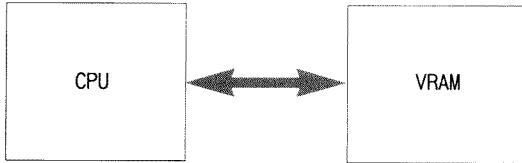
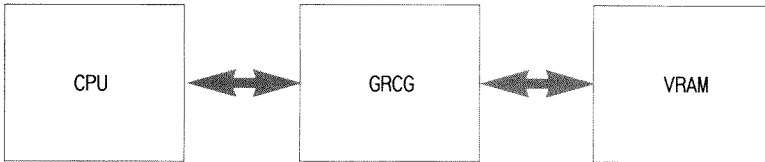
CPU　⇔　VRAM

●GRCG ONの場合

図4-1 GRCG OFF／ONの場合のCPUとVRAMの関係

表4-1　GRCGのI/Oポート

リード／ライト	I/O アドレス	機　　能	データ D7 D6 D5 D4 D3 D2 D1 D0
ライト	7CH	GRCGモードレジスタの書き込み	＊1
	7EH	GRCGタイルレジスタの書き込み	← GRCGタイルレジスタ →

*1　GRCGモードレジスタの形式（I/Oアドレス7CH）

GRCGを使うときには，まず，I/Oアドレス7CHのGRCGモードレジスタにGRCGのモードを指定する値を出力します．このモード指定でGRCG ONを指定する（CGビットを1にする）と，GRCGがONにされ，CPUとVRAMの間にGRCGが介在するようになります．すると，CPUから見ると，A8000H，B0000H，B8000H，E0000Hの4箇所に4プレーンあったVRAMがすべて同等になり，それら4箇所のプレーンのどれに書いても同じ結果になるようになります．つまり，実質的には，CPUから見るとVRAMが1プレーンだけになってしまったように見えるわけです．

その1プレーンだけのように見えているVRAMに対して書き込みが行われた場合に，実際にどのプレーンに対して書き込みが行われるかは，GRCGモードレジスタのP0～P3ビットで指定します．各ビットが0のときにそのプレーンに対する書き込みが有効になります．これらのビットはそれぞれ独立に指定できますから，1回の書き込みで4枚あるプレーンの任意のプレーンに対して同時に書き込みができることになります．P0～P3をすべて0にしてしまえば，1回の書き込みで4枚すべてのプレーンに書き込みを行うことも可能です．

このとき，書き込み対象になっているVRAMに実際に書き込まれる値は，CPUから書き込まれた値そのままではありません．GRCGは，CPUから書き込まれた値と，VRAMに元からあった値，それにI/Oアドレス7EHを介して書き込まれたタイルレジスタの値の3つの値を元にして演算を行い，その結果をVRAMに書き込みます．そのタイルレジスタの値は，4枚のプレーンそれぞれについて異なる値を指定することができます．

それぞれのプレーンに対するタイルレジスタの値を書き込むには，次のようにします．まず，GRCGのモードレジスタ（I/Oアドレス7CH）に値を出力してGRCGを有効にしてから，I/Oアドレス7EHに値を書き込んでいくと，最初に7EHに書き込んだ値がプレーン0に対するタイルレジスタ，次に書き込んだものがプレーン1に対するタイルレジスタ･･･となります．したがって，4プレーン分のタイルレジスタを書き込むときには，GRCGを有効にしてから，プレーン0，1，2，3に対するタイルレジスタを順にI/Oアドレス7EHに出力するようにします．

こうして指定されたタイルレジスタは当然，サイズは8ビットです．VRAMに対する16ビットアクセスが生じたときには，上位8ビットと下位8ビットに同じタイルレジスタの値が使われます．

以上が，GRCGについての基本事項ですが，GRCGにはVRAMに対する読み込みか書き込みか，あるいはVRAMに書き込むデータをどのように作成するかによって3つのモードが存在していて，実際にGRCGを使うときには用途によってそれらのうちの適当なモードを選択してやる必要があります．そのモード選択はGRCGモードレジスタのRMWビットで行います．以下，それらGRCGの3つのモードについて順に解説していきます．

●TDWモード

GRCGモードレジスタ（I/Oアドレス7CH）のCGビットを1，RMWビットを0に指定してからCPUがVRAMへの書き込みを行うと，GRCGはTDWモードの動作をします．

TDWモードでは，CPUから書き込まれたデータと，VRAMに元からあったデータはまったく無視されてしまい，タイルレジスタの値がそのままVRAMに書き込まれます．したがって，このTDWモードが有効な場合というのは，VRAMにある一定の値をずっと書き込んでいくような場合（たとえば画面クリアなど）に限られます．

ただし，このTDWモードでは，GRCGはCPUからもVRAMからもデータを読み込まなくてよいので，書き込み速度は非常に高速になります．TDWモードでの書き込み速度はRMWモード（後述）のほぼ2倍で，しかも大部分の機種では，GRCGを使わないで1プレーン書き込みをする速度よりも，TDWモードで複数プレーン同時書き込みをする速度の方が高速になるので，TDWモードを使えばGRCGを使わない場合の4倍以上の書き込み速度を実現することができます（4プレーン同時書き込み時）．

さて，TDWモードの具体的な使い方の例として，C言語でVRAMをクリアする場合を考えてみましょう．まず，GRCG ON，TDWモード，4プレーン同時書き込みを指定するために，

```
outportb (0x7c, 0x80) ;
```

とします．次に，4枚のプレーンに対するタイルレジスタをI/Oアドレス7EHに設定しますが，今は画面クリアですから，各プレーンに書き込むデータはすべて00Hでよいので，

```
for (i = 0; i < 4; i++)
  outportb (0x7e, 0x00) ;
```

とします．

それから，VRAMに対する書き込みを行っていきます．GRCG ONのときは4枚のプレーンがすべて同等なので，プレーン0（セグメントアドレスA800H）に対して書き込みを行うことにします．で，TDWモードではCPUからどんな値を書き込んでも同じことなので，とりあえず0000Hを書き込むことにすると，VRAM1プレーンのサイズは8000Hバイトなので，

```
for (i = 0; i < 0x8000; i+=2)
  poke (0xa800, i, 0x0000) ;
```

でクリアが実行されます．

そして，GRCGによる作業が終了したら必ず，以後CPUが普通にVRAMにアクセスすることができるようにするために，

```
outportb (0x7c, 0x00) ;
```

としてGRCGをOFFにするようにします．以上の手順をまとめてプログラム化したものをサンプルプログラムとして付けておきます．

このように，1枚のプレーンを扱うときとほとんど同じようにするだけで4枚のプレーンを同時に扱うことができる，というところがGRCGの大きな特長です．

■サンプルプログラム

GRCGを使ってグラフィック画面をクリアします．

```
#include <stdio.h>
#include <dos.h>

void main( void )
{
```

```
    unsigned i;

    outportb( 0x7c, 0x80 );
    for ( i = 1; i <= 4; i++ ) outportb( 0x7e, 0x00 );
    for ( i = 0; i < 0x8000; i += 2 ) {
        poke( 0xa800, i, 0 );
    }
    outportb( 0x7c, 0x00);
}
```

●RMWモード

GRCGモードレジスタ（I/Oアドレス7CH）のCGビット，RMWビットをともに1にしてからCPUがVRAMへの書き込みを行うと，GRCGはRMWモードの動作をします．

RMWモードでは，CPUからVRAMへの書き込みが行われると，CPUから書き込んだデータのビットが“1”の部分にのみタイルレジスタのデータが書き込まれ，“0”の部分は書きかえられないで，VRAMに元からあったデータがそのまま残されます（図4-2参照）．

CPUデータ	1	1	1	1	0	0	0	0
タイルレジスタ	1	0	1	0	0	1	1	0
	↓	↓	↓	↓	↓	↓	↓	↓
書き込まれるデータ	1	0	1	0	×	×	×	×

（×の部分はVRAMに元からあったデータがそのまま残される）

図4-2　RMWモードでのGRCGの動作

つまりこのモードは，ビット（ドット）単位の任意の領域にタイルレジスタの内容を書き込むもので，CPUからのデータはその書き込み領域の指定に使うわけです．

このRMWモードは，TDWモードと比べてかなり応用範囲が広く，たとえば次のような用途に使われます．

1）特定領域（四角形の内部など）の特定パターンでの塗りつぶし
2）1点の描画
3）グラフィック文字の描画

1）の場合は，書き込みたいパターンをタイルレジスタに入れておき，書き込みを行いたい部分だけを1にしたデータをCPUから書き込んでいくようにします．

2）の場合は，書き込みたい点のパレット番号の，相当するビットが0のプレーンのタイルレジスタには00Hを，相当するビットが1のプレーンのタイルレジスタにはFFHを書き込み，点を打ちたい部分のビットのみを1にしたデータをCPUから書き込むようにします．たとえば，（323，200）の位置にパレット番号0AH（1010B）の点を打ちたいとすると，GRCGをRMWモードにしてから，プレーン0，1，2，3のタイルレジスタをそれぞれ00H，FFH，00H，FFHに設定し，セグメントアドレス

A800H（プレーン0），オフセットアドレス200×80＋323/8=3EA8Hに，80Hを323%8=3回左シフトした値（10H）を書き込みます（サンプルプログラム参照）．

3）の場合は，2）の1点の描画のときと同じようにして書き込む文字のパレット番号によってタイルレジスタを設定し，文字パターンをCPUから書き込んでいきます．そうすると，文字パターンが1の部分にだけタイルレジスタによって指定した色が書き込まれ，それ以外の部分は書きかえられないので，自然な感じで文字を書き込むことができます．

このほか，ゲームなどでの重ね合わせ処理にこのRMWモードが用いられることがあるようですが，それは速度という観点からするとあまり好ましくありません．それは，次のような理由からです．

まず，GRCGでは書き込むデータをタイルレジスタに設定しますが，タイルレジスタはビット幅が8ビットしかないので，重ね合わせ処理においては，VRAMに対する16ビットアクセスはできず，8ビットずつのアクセスしかできません．また，RMWモードではいったんVRAMのデータを読み込んでからVRAMへの書き込みを行わなければならないので，RMWモードでのVRAMアクセスは通常のVRAMアクセスの2倍近くかかってしまいます．さらに，重ね合わせ処理を行うには毎回タイルレジスタを書きかえる必要がありますが，タイルレジスタのアクセスタイムは非常に長く，VRAMへのアクセスタイムの2倍以上もかかってしまう機種も珍しくありません．

以上のことから，VRAM4プレーンの16ドットの領域に重ね合わせ処理を行うのに必要な時間を通常のVRAMへの1回アクセスの時間を単位にして表してみると，8ビットずつしかアクセスできませんから2回アクセス，1アクセスに2倍かかりますからそれの2倍で，それに加えてタイルレジスタ4プレーン分を2回書き込まなくてはならなくて，1回の設定に1アクセスの2倍かかりますから結局，

$$2\times2+4\times2\times2=20 \text{（回分）}$$

の時間がかかることになります．一方，GRCGを使わなければ，普通にVRAMを読み込み，それを処理してから書き込むという操作を4プレーン分するわけですから，

$$2\times4=8 \text{（回分）}$$

となります．実際には，このほかに1プレーンにつき2回の論理演算処理が必要ですが，VRAMアクセスには非常に時間がかかるので，その処理にかかる時間は無視できる程度のものです．結局，重ね合わせ処理については，タイルレジスタが遅い大部分の機種ではGRCGを使わない方が2.5倍近くも高速であることになります．もっとも，まれにタイルレジスタがVRAMと同等以上のアクセスタイムでアクセスできる機種もありますが，そういう機種でもGRCGを使ってもそれほど速くはなりません．結局，GRCGは一般的な重ね合わせ処理，さらにいうとタイルレジスタを毎回書きかえなければならないような処理には向いていない，ということになります．

■サンプルプログラム

GRCGを使った点描画関数，漢字描画関数を用いてグラフィック画面を作ります．

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>
```

```
void setpal( int, int, int, int );
void pset( int, int, int );
void grcolor( int );
void putkan( int, int, char *, int );
void inpkan( int, char *);

void main( void )
{
    int i, x, y, cl, kcod;
    char kanpt[32];

    clrscr();
    outportb(0x68, 0x08);           /* 400ラインモード指定 */
    outportb(0xa2, 0x4b);
    outportb(0xa0, 0);
    outportb(0x6a, 1);              /* 16色モード */
    outportb(0xa2, 0x0d);           /* グラフィック画面　表示開始 */
    for (i = 0; i <= 14; i++) setpal(i, 0, i+1, 0);  /* パレットセット */
    setpal(15, 12, 12, 12);
    outportb(0x7c, 0xc0);                    /* GRCG RMWモード */

    for ( y = 0; y < 400; y++ ) {            /* 背景描画 */
        for ( x = 0; x < 640; x++ ) {
            cl = ( x + y ) / 4 % 15;
            pset( x, y, cl );
        }
    }

    outportb(0x68, 0x0b);            /* KCGモード=ビットマップモード */
    kcod = 0x3020;
    for ( y = 0; y < 400; y += 16 ) {        /* 文字描画 */
        for ( x = 0; x < 640; x += 16 ) {
            inpkan( kcod, kanpt );
            putkan( x, y, kanpt, 15 );
            kcod++;
        }
    }

    outportb(0x7c, 0x00);            /* GRCG OFF */
    outportb(0x68, 0x0a);            /* KCGモード=コードアクセスモード */
}

void setpal( int pnum, int gbr, int rbr, int bbr )  /* パレット設定関数 */
{
    outportb(0xa8, pnum);
    outportb(0xaa, gbr);
    outportb(0xac, rbr);
    outportb(0xae, bbr);
```

```
}

void grcolor( int cl )              /* GRCGカラーセット関数 */
{
   int i;

   for ( i = 1; i <= 4; i++ ) {
      if ( ( cl & 1 ) == 0) outportb( 0x7e, 0x00);
                          else outportb( 0x7e, 0xff);
      cl = cl >> 1;
   }
}

void pset( int x, int y, int cl )              /* 点打ち関数 */
{
   int vadd, vdta;

   grcolor( cl );
   vadd = y * 80 + x / 8;
   vdta = 0x80 >> (x % 8);
   pokeb( 0xa800, vadd, vdta);
}

void putkan( int x, int y, char kanpt[], int cl )      /* 漢字描画関数 */
{
   unsigned i, vadd, kpat;

   grcolor( cl );
   vadd = y * 80 + x / 8;
   for ( i = 0; i < 16; i++ ) {
      kpat = kanpt[i * 2] + kanpt[i * 2 + 1] * 0x100;
      poke(0xa800, vadd, kpat);
      vadd += 80;
   }
}

void inpkan( int jiscod, char pbuf[] )    /* 文字パターン読み出し関数 */
{
   int i, ploc;

   outportb( 0xa1, jiscod % 0x100 );           /* 文字コード下位バイト指定 */
   outportb( 0xa3, jiscod / 0x100 - 0x20 );    /* 文字コード上位バイト指定 */
   for ( i = 0; i < 32; i++ ) {
      ploc = i / 2;
      if ( (i % 2) == 0 ) ploc |= 0x20;
      outportb( 0xa5, ploc );                  /* 読み出し位置指定 */
      pbuf[i] = inportb( 0xa9 );               /* パターンの読み出し */
   }
}
```

●TCRモード

GRCGモードレジスタ（I/Oアドレス7CH）のCGビットを1，RMWビットを0に指定してからCPUがVRAMからの読み込みを行うと，GRCGはTCRモードの動作をします．

TCRモードでCPUがVRAMからの読み込みを行うと，GRCGは読み込んだVRAMからのデータとタイルレジスタの値をビット単位で比較し，有効プレーンすべてについてVRAMからのデータとタイルレジスタの値が一致したビットを1，1プレーンでもそれらが一致しなかったビットを0とした数値をCPUに返します．つまり，4バイトのタイルレジスタが横8ドット分のパレット番号を表していると見た場合に，タイルレジスタが表すパレット番号とVRAMにあるデータが表すパレット番号が一致する部分のビットは1，一致しない部分のビットは0となるのです（図4-3参照）．

図4-3　TCRモードでのGRCGの動作

このTCRモードを使うと効率的に行える処理としては，特定のパレット番号を持つ領域を抽出するというものがあります．この処理をTCRモードを使って行うには，抽出したい領域のパレット番号をRMWモードのときと同様にしてタイルレジスタに設定し，セグメントアドレスA800Hから1プレーン分のデータを読み込みます．そうすると，タイルレジスタに設定されたパレット番号を持つ領域のビットのみが1であるようなデータが得られます．

■4-1-2 EGC

EGC（Enhanced Graphic Charger）というのは「高機能のグラフィックチャージャ」というような意味ですが，EGCはその名の通り従来のGRCGの機能を大幅に拡張した強力なグラフィックチャージャです．その主な機能としては，次のようなものがあります．

1）4（8*)枚のプレーンにハードウェア的に同時アクセス可能
2）3つの値の間のあらゆる論理演算が行えるラスタオペレーション（ROP）機能
3）ビット単位のシフト・書き込み領域指定が可能
4）VRAM上の任意領域の高速ブロック転送機能
5）GDCからの描画制御が可能
6）GRCG互換モードあり

＊256色ボード搭載のH98のみ．

EGCに関係するI/Oポートを表4-2に示しておきますが，このほかにもI/Oアドレス04A0H～04AEHにEGCの動作を制御するI/Oポートが存在しています．

表4-2　EGCに関係するI/Oポート

リード／ライト	I/0 アドレス	機　　能	データ D7 D6 D5 D4 D3 D2 D1 D0
ライト	7CH	GRCGﾓｰﾄﾞﾚｼﾞｽﾀの書き込み	*1
	6AH	ﾓｰﾄﾞﾌﾘｯﾌﾟﾌﾛｯﾌﾟ2の書き込み	*2

*1 GRCGモードレジスタ（I/Oアドレス7CH）の形式（EGC拡張モードの場合）

D7	D6	D5	D4	D3	D2	D1	D0
CG	0	0	0	0	0	0	0

CG：EGCのON／OFF指定
（0：EGC　OFF，
1：EGC　ON）

*2 モードフリップフロップ2（I/Oアドレス6AH）に出力する値と動作の関係
（EGCに関係するもの）

出力する値	意　　味	解　　説
04H	GRCG互換モード	EGCの動作モードの選択（EGCモード変更可のときのみ有効）
05H	EGC拡張モード	
06H	EGCモード変更不可	EGCの動作モード変更の可否の選択
07H	EGCモード変更可	

実際にEGCを使うには，まず，GRCGのときと同じようにI/Oアドレス7CHのCGビットを1にして，グラフィックチャージャをONにします．ただし，それだけではEGCはGRCG互換モードの動作をするので，GRCG相当の機能しか使えません．EGCの拡張機能を使うときには，モードフリップフロップ2（I/Oアドレス6AH）を制御して，EGC拡張モードを選択します．具体的には，C言語なら，

```
outportb( 0x7c, 0x80 );
outportb( 0x6a, 0x07 );
outportb( 0x6a, 0x05 );
```

としてやります．

そうすると，GRCGのときと同じようにCPUとVRAMの間にEGCが介在するようになり，CPUから見ると4個所のVRAMがすべて同等にみえるようになります．

それから，I/Oアドレス04A0H～04AEHまでのI/Oポートを制御してEGCの動作を設定してやります．そこで，以下，それらのI/Oポートについての詳しい解説を行いたいと思いますが，実は，EGCの仕様や使用法はメーカーによって公開されていませんので，以下に示すのは著者が独自に解析した結果，こうではないかと推定したものです．したがって，それらがすべて正しいとはいいきれませんし，説明に使ったパラメータ名なども著者が勝手につけたもので正式名称ではない，ということを了解しておいてください．

なお，以下で解説するI/Oポートはすべて16ビットポートです．また，これらのI/OポートはGRCGモードレジスタのCGビット=1で，かつEGC拡張モードであるときにしか変更できないので注意してください．

1 I/Oアドレス04A0H（アクセスプレーンの指定）

各ビットの意味

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	P7	P6	P5	P4	P3	P2	P1	P0

Pn：プレーンnへのアクセスの有効／無効の指定
（0でアクセス有効，1でアクセス無効）

EGCの各プレーンへのアクセスの有効／無効を指定します．ここに0を設定したプレーンすべてに対して，ハードウェア的に同時アクセスが行われます．なお，P4～P7は256色ボードを搭載したH98でのみ有効です．

設定例　ノーマル98で4プレーン同時アクセスを行うようにするには，

```
outport( 0x04a0, 0xfff0 );
```

また，256色ボード搭載のH98で8プレーン同時アクセスを行うには，

```
outport( 0x04a0, 0xff00 );
```

とします（C言語の場合）．

2 I/Oアドレス04A2H（パターンデータおよびリードプレーンの指定）

各ビットの意味

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	←PC→		0	0	←RPN→			1	1	1	1	1	1	1	1

PC：ラスタオペレーションのパターンデータに使うデータの指定

PC=00B：パターンレジスタ

PC=01B：バックグラウンドカラー

PC=10B：フォアグラウンドカラー

PC=11B：設定不可

ラスタオペレーションのパターンデータとして何を使うかを指定します．特定色での描画にはフォア（バック）グラウンドカラー，特定パターンの描画にはパターンレジスタを使うのが便利です．

RPN：単一プレーンリード時のリードプレーン番号の指定

リードモードが単一プレーンリードの場合に，リードするプレーンのプレーン番号を指定します．ノーマル98では0（000B）～3（011B）まで，256色ボード搭載のH98では0（000B）～7（111B）までの値を指定することができます．

設定例 パターンデータをフォアグラウンドカラー，リードプレーンはプレーン2とするには，PC=10B，RPN=010Bとすればよいので，

```
outport( 0x04a2, 0x42ff );
```

とします（C言語の場合）．

3 I/Oアドレス04A4H（リード／ライトモードの指定）

各ビットの意味

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	RM	←WM→		SD	←PS→		R7	R6	R5	R4	R3	R2	R1	R0

RM：VRAMリード時の動作モードの指定

RM=0：単一プレーンリード

RM=1：コンペアリード

CPUがVRAMからのデータリードを行ったときの動作を指定します．

単一プレーンリードが指定された場合は，RPN（I/Oアドレス04A2H）で指定されたプレーン番号のプレーンがリードされます．

コンペアリードが指定された場合は，フォアグラウンドカラーと同じ色の領域のビットのみが1になったデータがリードされます．このコンペアリードを行う場合には，必ずPC（I/Oアドレス04A2H=10B（フォアグラウンドカラー指定）と設定するようにします．

WM：VRAMライト時の動作モードの指定

WM=00B：CPUからのデータをそのまま書き込む
WM=01B：ラスタオペレーションの結果をシフタ経由で書き込む
WM=10B：パターンデータをシフタ経由で書き込む
WM=11B：？

CPUがVRAMへのデータライトを行ったときの動作を指定します．
WM=00Bとした場合には，CPUから書き込まれた値を，ラスタオペレーション，ビットシフトなどの操作を行うことなくそのままVRAMに書き込みます．

WM=01Bとすると，ソースデータ，パターンデータ，VRAMデータの3つの値からラスタオペレーションを行って作成した値を，ビットシフトを行ってからVRAMに書き込みます．ソースデータとしてはCPUからのデータあるいは直前にVRAMリードしたときのリードデータを，パターンデータとしてはパターンレジスタあるいはフォア（バック）グラウンドカラーを使うことができます．VRAMデータは常に書き込まれる前にVRAMにあった値が使われます．どのようなラスタオペレーションを行うかは，同じI/OアドレスのR0～R7で指定します．また，ビットシフトはDAD(I/Oアドレス04ACH)に指定されたビット幅だけ行われます
WM=10Bとすると，パターンデータの値を,DADで指定されたビット幅だけビットシフトしてからVRAMに書き込みます．ラスタオペレーションを行うときと同様に，パターンデータとしてはパターンレジスタまたはフォア（バック）グラウンドカラーを指定することができます．

SD：ラスタオペレーションのソースデータとするデータの指定

SD=0：ソースデータとしてVRAMリードをしたときのリードデータを使う
SD=1：ソースデータとしてCPUからのデータを使う

ラスタオペレーションを行うときのソースデータとして何を使うかを指定します．
SD=0とすると，直前にVRAMリードを行ったときのリードデータがソースデータとして使われます．
SD=1とすると，CPUから書き込まれたデータがソースデータとして使われます．

PS：パターンレジスタのセット方法の指定

PS=00B：パターンレジスタは変化しない
PS=01B：ソースデータと同じものがセットされる
PS=10B：VRAMデータと同じものがセットされる
PS=11B：変化しない？

パターンレジスタの変更方法を指定します．
PS=00Bとするとパターンレジスタの内容は変化しません．
PS=01Bとすると，SD=0のときVRAMリード時にVRAMからのリードデータが

セットされ，SD=1のときVRAMライト時にCPUからのデータがセットされるので，結局ソースデータと同じものがセットされることになります．

PS=10Bとすると，VRAMライト時にVRAMに元からあったデータがセットされるので，VRAMデータと同じものがセットされることになります．つまり，PS=01BまたはPS=10Bとすると，パターンデータとしてパターンレジスタを指定した場合には実質的にソースデータとVRAMデータの2値間のラスタオペレーションとなることになります．

なお，いずれの場合もセットされるパターンレジスタはI/Oアドレス04A0Hで指定して有効にしたプレーンに対応するもののみです．

R0〜R7：ラスタオペレーションの演算指定コード（ROPコード）の指定

ラスタオペレーションを行うときに，どのような論理演算を行うかを指定します．ここに指定した値と行われる論理演算の関係は，次のようなものです．

今，ソースデータをS，VRAMデータをV，パターンデータをPと表すことにし，・はAND（論理積），＋はOR（論理和），$\overline{X}$はNOT（反転）を表すとすると，行われる論理演算はVRAMに書き込まれるデータをDとして，

$$D=R7\cdot S\cdot V\cdot P+R6\cdot S\cdot V\cdot \overline{P}+R5\cdot S\cdot \overline{V}\cdot P+R4\cdot S\cdot \overline{V}\cdot \overline{P}$$
$$+R3\cdot \overline{S}\cdot V\cdot P+R2\cdot \overline{S}\cdot V\cdot \overline{P}+R1\cdot \overline{S}\cdot \overline{V}\cdot P+R0\cdot \overline{S}\cdot \overline{V}\cdot \overline{P}$$

となります．

たとえば，GRCGのRMWモードと同じ論理演算をさせるには，まず，CPUからのデータをソースデータにするためにSD=1としたうえで，S（ソースデータ）=1ならP（パターンデータ）の値を，S=0ならV（VRAMデータ）の値をそのまま書き込めばよいので，演算式は，

$$D=\underbrace{S\cdot V\cdot P+S\cdot \overline{V}\cdot P}_{S=1でPをそのまま}+\underbrace{\overline{S}\cdot V\cdot P+\overline{S}\cdot V\cdot \overline{P}}_{S=0でVをそのまま}$$

となるので，R7=R5=R3=R2=1，R6=R4=R1=R0=0，つまりROPコードとしてACHを指定してやればいいことになります．

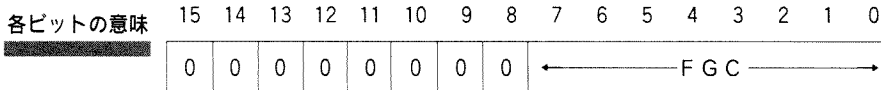
設定例

04A4Hに出力する値	動作
0CACH	GRCGのRMWモード相当
1000H	GRCGのTDWモード相当
0CC0H	CPUデータとVRAMデータのANDを書き込む
0C3CH	CPUデータとVRAMデータのXORを書き込む
0400H	CPUデータをそのまま書き込む
1500H	CPUデータをシフトして書き込む
0DCCH	パターンレジスタにCPUデータをセットする
1100H	VRAM間ブロック転送（ROPなし）を行う場合
08xxH	VRAM間ブロック転送（ROPあり）を行う場合

上記の値のときリードを行うと単一プレーンリードになる．コンペアリードを行うときは上記の値＋2000Hを指定する．

4 I/Oアドレス04A6H（フォアグラウンドカラーの指定）

各ビットの意味

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	←FGC→							

FGC：フォアグラウンドカラーの指定

ROPのパターンデータなどとして使われるフォアグラウンドカラーを指定します．ノーマル98では0～15，256色ボード搭載のH98では0～255の数値を指定することができます．

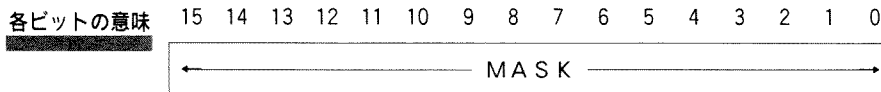
設定例 フォアグラウンドカラーをパレット番号12の色に設定するには，

```
outport( 0x04a6, 12 );
```

とします（C言語の場合）．

5 I/Oアドレス04A8H（マスクレジスタの設定）

各ビットの意味

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
←MASK→															

MASK：設定するマスクレジスタの値

（ビットが0の部分=書き込みが無効，ビットが1の部分=書き込みが有効）

全プレーンに対して共通に働くマスクレジスタを設定します．このマスクレジスタのビットが0の部分には常にVRAMに元からあったデータが残され，書き込みはマスクレジスタのビットが1の部分にのみ行われます．たとえば，MASK=AAAAHとすれば書き込みは1ビットおきの領域にのみ行われます．

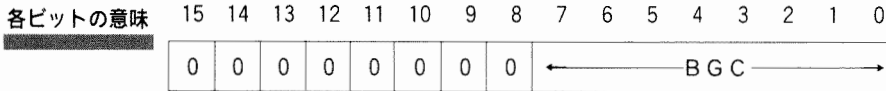
設定例　すべての部分への書き込みを有効にするには，MASKの全ビットを1にすればよいので，

```
outport( 0x04a8, 0xffff );
```

とします（C言語の場合）．

6 I/Oアドレス04AAH（バックグラウンドカラーの指定）

各ビットの意味

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	← BGC →							

BGC：バックグラウンドカラーの指定

ROPのパターンデータなどとして使われるバックグラウンドカラーを指定します．ノーマル98では0～15，256色ボード搭載のH98では0～255の数値を指定することができます．なお，バックグラウンドカラーは，コンペアリードのときの比較色としては使えないようです．

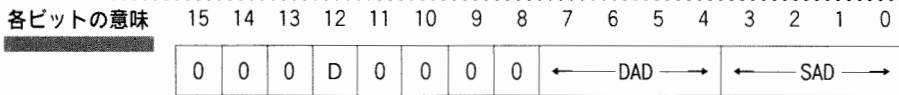
設定例　バックグラウンドカラーをパレット番号0の色に設定するには，

```
outport( 0x04aa, 0 );
```

とします（C言語の場合）．

7 I/Oアドレス04ACH（ビットアドレスおよびブロック転送方向の指定）

各ビットの意味

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	D	0	0	0	0	← DAD →				← SAD →			

D：ブロック転送方向の指定

（0：アドレスの低い方から転送，1：アドレスの高い方から転送）

VRAM間のブロック転送を行うときの転送方向を指定します．設定すべき値はCPUのディレクションフラグの値によって異なりますが，通常は0を設定します．

DAD：デスティネーションビットアドレスの指定

転送先のビットアドレスを指定します．VRAM間転送のときはもちろんですが，ROPの結果やパターンデータの書き込みのときにもここで指定したビットアドレスは有効です．したがって，ROPの結果などをシフトを行わないでそのまま書き込みたいときには，DAD=0を指定します．

SAD：ソースビットアドレスの指定

転送元のビットアドレスを指定します．このSADとDAD，それに次に出てくるBLNを指定してからブロック転送を行えば，ドット単位で任意の領域をVRAM上の任意の位置に転送することができます．

設 定 例 転送方向は低い方から，ソースビットアドレスを6，デスティネーションビットアドレスを3にするには，

```
outport( 0x04ac, 0x0036 );
```

とします（C言語の場合）．

8 I/Oアドレス04AEH（書き込み領域のビット長の指定）

各ビットの意味

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	←					BLN						→

BLN：書き込み領域のビット長−1の指定

書き込みを行う際に，実際に書き込まれる領域のビット長−1を指定します．ここで指定したビット長から溢れた部分には書き込みは行われず，VRAMに元からあったデータがそのまま残されます．このビット長は，すべての書き込み（CPUデータ，ROPの結果，パターンデータ）に対して有効なので，書き込むビット数に制限をつけたくないときはDAD（デスティネーションビットアドレス）=0かつBLN=15（ビット長16ビット）を指定するようにします．

設 定 例 書き込み領域のビット長を16ビットに設定するには，

```
outport( 0x04ae, 0x000f );
```

とします（C言語の場合）．

以上のI/Oポートを設定してから，CPUからプレーン0（セグメントアドレスA800H）に対する書き込み・ブロック転送などを行えば，実際の描画が行われます．

そして，EGCによる描画が終了したなら，必ずEGCをGRCG互換モードに戻し，EGCモード変更を禁止し，GRCG OFFにしておくようにします．具体的には，

```
outportb( 0x6a, 0x04 );
outportb( 0x6a, 0x06 );
outportb( 0x7c, 0x00 );
```

とします．

■サンプルプログラム

EGCを使った点描画関数を用いて図形を描画します．

```
#include <stdio.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
```

```
void pset( int, int, int);

void main( void )
{
   int i, x, y, cl, gpar;

   clrscr();
   outportb(0x6a, 1);                   /* 16色モード */
   outportb(0xa2, 0x4b);                /* GDC CSRFORMコマンド */
   outportb(0xa0, 0);                   /* L/R = 1 */
   outportb(0x68, 8);                   /* モードF/F1 */
   outportb(0xa2,0xd);

   for (i = 0; i < 16; i++) {
      outportb(0xa8, i);                /* 16色モードのパレット設定 */
      outportb(0xaa, i);
      outportb(0xac, 0);
      outportb(0xae, i);
   }

   outportb(0x6a,7);                 /* EGCモード変更可 */
   outportb(0x6a,5);                 /* EGC拡張モード */
   outportb(0x7c,0x80);              /* EGC  ON */
   outport(0x04a0,0xfff0);           /* 4プレーン同時 */
   outport(0x04a2,0x40ff);           /* パターンデータ = フォアカラー */
   outport(0x04a4,0x0cac);           /* ROP, RMW相当 */
   outport(0x04a8,0xffff);           /* マスクなし */
   outport(0x04ac,0x0000);           /* シフトなし */
   outport(0x04ae,0x000f);           /* ビット長=16 */
   for (y = 0; y < 400; y++) {       /* 図形描画 */
      for (x = 0; x < 640; x++) {
         gpar = 48.0 * sin( x / 30.0 );
         cl = ( x + y + gpar ) / 4 % 16;
         pset(x, y, cl);
      }
   }
   outportb(0x7c,0);                 /* EGC  OFF */
   outportb(0x6a,4);                 /* GRCG互換モード */
   outportb(0x6a,6);                 /* EGCモード変更不可 */
}

void pset(int x, int y, int cl)      /* 点描画関数 */
{
   int vadd, vdta;

   vadd = y * 80 + x / 8;
   vdta = 0x80 >> (x % 8);
   outport( 0x04a6, cl );            /* フォアカラーセット */
   pokeb( 0xa800, vadd, vdta);
}
```

●GDCからの描画制御

GRCGはGDCからの描画は制御することができませんでしたが，EGCはGDCからの描画も制御することができるようになっています．つまり，EGCは，GDCがVRAMに対して書き込みを行ったときにも，CPUが書き込みを行ったのと同等の処理（4プレーン同時書き込み，ラスタオペレーションなど）を行うことができるのです．

GDCは，描画を行うときにはまずVRAM上のデータを読み込み，それと書き込む図形との間で論理演算（REPLACE，COMPLEMENT，CLEAR，SETのうちのいずれか）を行ってからそれをVRAMに書き込んでいます．しかし，EGCがONになっていると，GDCが書き込みを行う前にVRAMから読み込みを行うと，VRAMに何が書いてあっても常に0が読み込まれます．つまり，GDCから見るとVRAMには何も書いてないように見えるわけです．そのため，GDCが，読み込んだ0と書き込む図形の間で上記4つの論理演算を行っても，CLEARは常にすべてのビットが0になったデータを書き出すので意味がなく，REPLACE，COMPLEMENT，SETはすべて同じ結果（書き込む図形の部分のビットのみが1になったデータ）を書き出します（2−7−4．グラフィックGDC参照）．その書き出されるデータをEGCを介して書き込むと，通常，GDC単独でのSETのときと同等の結果を与えます．ですから，GDCからEGCを介しての描画を行うときには，GDCのドット修正モードはSETにしておくようにします．

そのうえで，たとえば通常の直線の4プレーン同時描画が行いたいときには，EGCをGRCG互換モード（後述）のRMWモードにするか，あるいはEGC拡張モードでRMWモード相当のラスタオペレーションをするように設定し，タイルレジスタ（パターンデータ）で直線の色を指定してからGDCでプレーン0のみに対しての描画を行ってやります．この方法を使えば，8色モードにしか対応していないグラフィックBIOSを使って16色モードの描画を行うことも可能となります．

■サンプルプログラム

グラフィックBIOSとEGCを使って図形を描画します．

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>

void line(int, int, int, int, int);
void grcolor( int );

union bufs {
   char byte;
   int word;
};

union REGS inregs, outregs;
struct SREGS segregs;
union bufs buf[40];                        /* 80バイトのバッファ */

void main(void)
{
```

```
    int i, cl;

    clrscr();
    inregs.h.ch = 0xc0;                     /* 640×400ドット */
    inregs.h.ah = 0x42;                     /* 画面モードの設定 */
    int86(0x18, &inregs, &outregs);
    outportb(0x6a, 0x01);                   /* 16色モード */
    outportb(0xa6, 0);                      /* 表画面に描画 */
    inregs.h.ah = 0x40;                     /* グラフィック画面表示開始 */
    int86(0x18, &inregs, &outregs);
    for (i = 0; i < 16; i++) {              /* パレットの設定 */
        outportb(0xa8, i);
        outportb(0xaa, i);
        outportb(0xac, i);
        outportb(0xae, 0);
    }
    outportb(0x7c, 0xc0);                   /* GC  ON */
    outportb(0x6a, 0x07);                   /* モード変更可 */
    outportb(0x6a, 0x05);                   /* EGC拡張モード */
    outport(0x04a0,0xfff0);                 /* 4プレーン同時アクセス */
    outport(0x04a2,0x40ff);                 /* パターンデータ=フォアカラー */
    outport(0x04a4,0x0cac);                 /* RMWモード相当 */
    outport(0x04a8,0xffff);                 /* マスクなし */
    outport(0x04ac,0x0000);                 /* シフトなし */
    outport(0x04ae,0x000f);                 /* ビット長=16ビット */

    cl = 0;
    for (i = 0; i < 640; i += 5) {          /* 図形の描画 */
        line(0, 0, i, 399, cl % 16);
        cl++;
    }

    outportb(0x6a, 0x04);                   /* GRCG互換モード */
    outportb(0x6a, 0x06);                   /* モード変更不可 */
    outportb(0x7c, 0x00);                   /* GC  OFF */
}

void line(int x1, int y1, int x2, int y2, int cl)
{
    segregs.ds = FP_SEG(buf);
    inregs.x.bx = FP_OFF(buf);
    buf[0x02 / 2].byte = 3;                 /* 描画モード=SET */
    buf[0x08 / 2].word = x1;
    buf[0x0a / 2].word = y1;
    buf[0x16 / 2].word = x2;
    buf[0x18 / 2].word = y2;
    buf[0x20 / 2].word = 0xffff;            /* ラインスタイル=ノーマル */
    buf[0x28 / 2].byte = 1;                 /* 図形=直線 */
    outport(0x04a6, cl);                    /* カラーセット */
```

```
    inregs.h.ch = 0x80;
    inregs.h.ah = 0x47;                    /* 直線の描画 */
    int86x(0x18, &inregs, &outregs, &segregs);
}
```


●GRCG互換モード

GRCG互換モードにおいては，EGCはGRCGとほとんど同じ動作をします．具体的にどのような動作をするかはGRCGの項を参照してください．ただ1つ違うのは，GRCG互換モードにおいても，GRCGでは不可能だったGDCからの描画制御が可能である，ということです．

■サンプルプログラム

グラフィックBIOSとEGC（GRCG互換モード）を使って図形を描画します．

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>

void line(int, int, int, int, int);
void grcolor( int );

union bufs {
    char byte;
    int word;
};

union REGS inregs, outregs;
struct SREGS segregs;
union bufs buf[40];                        /* 80バイトのバッファ */

void main(void)
{
    int i, cl;

    clrscr();
    inregs.h.ch = 0xc0;                    /* 640×400ドット */
    inregs.h.ah = 0x42;                    /* 画面モードの設定 */
    int86(0x18, &inregs, &outregs);
    outportb(0x6a, 0x01);                  /* 16色モード */
    outportb(0xa6, 0);                     /* 表画面に描画 */
    inregs.h.ah = 0x40;                    /* グラフィック画面表示開始 */
    int86(0x18, &inregs, &outregs);
    for (i = 0; i < 16; i++) {             /* パレットの設定 */
        outportb(0xa8, i);
        outportb(0xaa, i);
        outportb(0xac, i);
        outportb(0xae, i);
    }
```

```
    cl = 0;
    for (i = 0; i < 640; i += 5) {          /* 図形の描画 */
        line(0, 0, i, 399, cl % 16);
        cl++;
    }
}

void line(int x1, int y1, int x2, int y2, int cl)
{
    segregs.ds = FP_SEG(buf);
    inregs.x.bx = FP_OFF(buf);
    buf[0x02 / 2].byte = 3;                 /* 描画モード=SET */
    buf[0x08 / 2].word = x1;
    buf[0x0a / 2].word = y1;
    buf[0x16 / 2].word = x2;
    buf[0x18 / 2].word = y2;
    buf[0x20 / 2].word = 0xffff;            /* ラインスタイル=ノーマル */
    buf[0x28 / 2].byte = 1;                 /* 図形=直線 */
    outportb(0x7c, 0xc0);                   /* GRCG ON */
    grcolor(cl);                            /* カラーセット */
    inregs.h.ch = 0x80;
    inregs.h.ah = 0x47;                     /* 直線の描画 */
    int86x(0x18, &inregs, &outregs, &segregs);
    outportb(0x7c, 0x00);                   /* GRCG OFF */
}

void grcolor( int cl )             /* GRCGカラーセット関数 */
{
    int i;

    for ( i = 1; i <= 4; i++ ) {
        if ( ( cl & 1 ) == 0) outportb( 0x7e, 0x00);
                             else outportb( 0x7e, 0xff);
        cl = cl >> 1;
    }
}
```

§4-2 メモリマップ

ノーマルモード

アドレス	内容	備考
FFFFFF FA0000		0A0000〜0FFFFFと同一の内容
F00000	リザーブ	
EFFFFF 100000	RAM空間	80286/80386（SX）CPU プロテクトモード時のみアクセス可
FFFFF バンク0F E80000	BIOS,BASIC ROM 96KB	
バンク0E E0000	グラフィックVRAM 32KB×2　プレーン3	
バンク0D C8000	システム用拡張ROM空間96KB （ユーザー使用不可）	
バンク0C C0000	ユーザー用拡張ROM空間32KB	
バンク0B B0000 A8000	グラフィックVRAM 96KB×2 プレーン2 プレーン1 プレーン0	
バンク0A A0000	テキストVRAM他	
バンク09 バンク08 80000 バンク07 バンク00 00000	RAM空間	

不揮発性メモリはA3FE2、A3FE4、A3FEA、A3FEE、A3FF2、A3FF4、A3FFA、A3FFEの8バイトがメモリスイッチとして使用される

ハイレゾモード

アドレス	内容	備考
FFFFFF FC0000		0A0000〜0FFFFFと同一の内容
F00000	リザーブ	
EFFFFF 100000	RAM空間	80286/80386（SX）CPU プロテクトモード時のみアクセス可
FFFFF バンク0F	BIOSROM　64KB	
バンク0E	システム用拡張ROM空間	
E0000		
バンク0D	グラフィックVRAM 128KB×4	
バンク0C C0000		
バンク0B B0000		
バンク0A A0000	RAMウィンドウ	
バンク09 90000		
バンク08 80000		
バンク07	RAM空間	
バンク06 60000		
バンク05		
バンク04 40000		
バンク03		
バンク02 20000		
バンク01		
バンク00 00000		

アドレス	偶数アドレス	奇数アドレス
A8000		
E6000	システム拡張用ROM空間 40KB	
E5000	ユーザー拡張ROM空間　4KB	
E4000	CGウィンドウ　4KB	
	不揮発性メモリ	
E2000	アトリビュート用 4KB	
E0000	ANK、日本語 表示用4KB	日本語表示用 4KB

E0000〜E4000：テキストVRAM

不揮発性メモリはA3FE2、A3FE4、A3FEA、A3FEE、A3FF2、A3FF4、A3FFA、A3FFE、の8バイトがメモリスイッチとして使用される

■I/Oマップ（ノーマルモード用）

番号	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	デバイス名
	I／Oアドレス																
1	×	×	×	×	0	0	×	×	0	0	0	0	0	×	A0	0	割り込みコントローラ8259相当（マスタ）
2	×	×	×	×	0	0	×	×	0	0	0	0	1	×	A0	0	割り込みコントローラ8259相当（スレーブ）
3	×	×	×	×	0	0	×	×	0	0	0	A3	A2	A1	A0	0	DMAコントローラ8237相当
4	×	×	×	×	0	0	×	×	0	0	1	0	×	×	×	0	カレンダ時計4990相当
5	×	×	×	×	0	0	×	×	0	0	1	0	×	A1	A0	1	DMAバンク
6	×	×	×	×	0	0	×	×	0	0	1	1	×	×	A0	0	RS-232Cインタフェース8251相当
7	×	×	×	×	0	0	×	×	0	0	1	1	×	A1	A0	1	システムポート8255相当
8	×	×	×	×	0	0	×	×	0	1	0	0	×	A1	A0	0	プリンタインタフェース8255相当
9	×	×	×	×	0	0	×	×	0	1	0	0	×	×	A0	1	キーボードインタフェース8251相当
10	×	×	×	×	0	0	×	×	0	1	0	1	×	×	A0	0	NMIコントロール
11	×	×	×	×	0	0	×	×	0	1	1	0	A2	A1	A0	0	GDCμPD7220A（テキスト）
12	×	×	×	×	0	0	×	×	0	1	1	1	A2	A1	A0	0	CRTC・GRCG
13	×	×	×	×	0	0	×	×	0	1	1	1	×	A1	A0	1	タイマコントローラ8253相当
14	×	×	×	×	×	×	×	×	1	0	0	0	0	0	A0	0	ハードディスクインタフェース
15	×	×	×	×	0	0	0	1	1	0	0	0	1	A1	A0	0	FM音源YM2203（YM2608）
16	×	×	×	×	×	×	×	×	1	0	0	0	1	A1	A0	1	ネットワークインタフェースボード
17	×	×	×	×	×	×	×	×	1	0	0	1	×	A1	A0	0	1MB　FDDコントローラ765相当
18	×	×	×	×	×	×	×	×	1	0	0	1	1	0	A0	1	GP－IBスイッチ
19	×	×	×	×	0	0	×	×	1	0	1	0	A2	A1	A0	0	GDCμPD7220A（グラフィック）
20	×	×	×	×	0	1	×	×	1	0	1	0	A2	A1	A0	0	EGC制御
21	×	×	×	×	1	0	0	1	1	0	1	0	A2	A1	A0	0	グラフィック制御
22	×	×	×	×	0	0	×	×	1	0	1	0	A2	A1	A0	1	KCG　ROM
23	×	×	×	×	×	×	×	×	1	0	1	1	A3	A2	A1	A0	通信制御アダプタ
24	×	×	×	×	×	×	×	×	1	0	1	1	A3	A2	A1	A0	RS-232C拡張インタフェース
25	×	×	×	×	×	×	×	×	1	0	1	1	1	1	1	0	1MB/640KB　FDD切換えインタフェース
26	×	×	×	×	×	×	×	×	1	1	0	0	1	A1	A0	0	640KB　FDDコントローラ765相当
27	×	×	×	×	×	×	×	×	1	1	0	0	A2	A1	A0	1	GP－IB7210相当
28	0	1	1	1	1	1	1	1	1	1	0	1	1	A1	A0	1	マウスコントローラ8255相当
29	0	0	1	1	1	1	1	1	1	1	0	1	1	A0	1	1	タイマコントローラ8253相当(ビープ音制御)
30	1	0	1	1	1	1	1	1	1	1	0	1	1	0	1	1	マウス割り込み時間間隔設定
31	×	×	×	×	0	0	×	×	1	1	1	1	0	A1	A0	0	CPU
32	×	×	×	×	0	0	×	×	1	1	1	1	1	A2	A1	A0	NDP

1．××××００××０００００×A0０　割り込みコントローラ8259相当（マスタ）
2．××××００××００００１×A0０　割り込みコントローラ8259相当（スレーブ）

I／O アドレス	リード／ ライト	機　　　　能
00H(マスタ) 08H(スレーブ)	リード ライト	ＩＲＲ，ＩＳＲまたは割り込みレベルリード ＯＣＷ２（ＥＯＩ），ＯＣＷ３，ＩＣＷ１ライト
02H(マスタ) 0AH(スレーブ)	リード ライト	ＩＭＲリード OCW1(IMR),ICW2,ICW3,ICW4ライト

3．××××００××０００A3A2A1A0０　ＤＭＡコントローラ8237相当

I／O アドレス	リード／ ライト	機　　　　能
０１Ｈ	リード／ライト	ＤＭＡチャネル０アドレスリード／ライト
０３Ｈ	リード／ライト	ＤＭＡチャネル０ワードカウントリード／ライト
０５Ｈ	リード／ライト	ＤＭＡチャネル１アドレスリード／ライト
０７Ｈ	リード／ライト	ＤＭＡチャネル１ワードカウントリード／ライト
０９Ｈ	リード／ライト	ＤＭＡチャネル２アドレスリード／ライト
０ＢＨ	リード／ライト	ＤＭＡチャネル２ワードカウントリード／ライト
０ＤＨ	リード／ライト	ＤＭＡチャネル３アドレスリード／ライト
０ＦＨ	リード／ライト	ＤＭＡチャネル３ワードカウントリード／ライト
１１Ｈ	リード	ＤＭＡステータスレジスタリード
	ライト	ＤＭＡコマンドレジスタライト
１３Ｈ	ライト	ＤＭＡリクエストレジスタライト
１５Ｈ	ライト	ＤＭＡシングルマスクレジスタビットライト
１７Ｈ	ライト	ＤＭＡモードレジスタライト
１９Ｈ	ライト	ＤＭＡクリアバイトポインタフリップフロップ
１ＢＨ	リード	ＤＭＡテンポラリレジスタリード
	ライト	ＤＭＡマスタクリア
１ＤＨ	ライト	ＤＭＡクリアマスタレジスタ
１ＦＨ	ライト	ＤＭＡオールマスクレジスタビットライト

4．×××××00××0010×××0　カレンダ時計4990相当

<table>
<tr><th>I／O
アドレス</th><th>リード／
ライト</th><th>機　　能</th></tr>
<tr><td>20H</td><td>ライト</td><td>
4990コマンドライト
<table>
<tr><th>D7</th><th>D6</th><th>D5</th><th>D4</th><th>D3</th><th>D2</th><th>D1</th><th>D0</th></tr>
<tr><td>×</td><td>×</td><td>DI</td><td>CLK</td><td>STB</td><td>C2</td><td>C1</td><td>C0</td></tr>
</table>
DI：シフトレジスタのデータ入力
CLK：シフトクロック
STB：ストローブ（0→1でモードセット）
<table>
<tr><th>グループ</th><th>C2 C1 C0</th><th>FUNCTION MODE</th></tr>
<tr><td rowspan="4">0</td><td>0 0 0</td><td>レジスタホールド DATA OUT=1Hz</td></tr>
<tr><td>0 0 1</td><td>レジスタシフト DATA OUT=[LSB]</td></tr>
<tr><td>0 1 0</td><td>タイムセット及びカウンタホールド
DATA OUT=[LSB]</td></tr>
<tr><td>0 1 1</td><td>タイムリード DATA OUT=0.5Hz</td></tr>
<tr><td rowspan="4">1</td><td>1 0 0</td><td>TP＝64Hz　SET</td></tr>
<tr><td>1 0 1</td><td>TP＝256Hz　SET</td></tr>
<tr><td>1 1 0</td><td>TP＝2048Hz　SET</td></tr>
<tr><td>1 1 1</td><td>拡張モード</td></tr>
</table>
</td></tr>
</table>

5．×××××00××0010×A1A01　DMAバンク

I／O アドレス	リード／ ライト	機　　能
23H	ライト	DMAチャネル2用バンクライト
25H	ライト	DMAチャネル3用バンクライト
27H	ライト	DMAチャネル0用バンクライト

6．××××00××0011××A00　RS-232Cインタフェース8251相当

<table>
<tr><th>I/O
アドレス</th><th>リード/
ライト</th><th>機　　能</th></tr>
<tr><td>30H</td><td>リード/ライト</td><td>受信データリード/送信データライト</td></tr>
<tr><td rowspan="2">32H</td><td>リード</td><td>ステータスリード
<table><tr><th>D7</th><th>D6</th><th>D5</th><th>D4</th><th>D3</th><th>D2</th><th>D1</th><th>D0</th></tr><tr><td>DSR</td><td>SYND</td><td>FE</td><td>OE</td><td>PE</td><td>TxE</td><td>RRDY</td><td>TRDY</td></tr></table>DSR：データセットレディ信号
SYND：同期検出（0：なし，1：あり）
FE：フレーミングエラー（0：なし，1：あり）
OE：オーバーランエラー（0：なし，1：あり）
PE：パリティエラー（0：なし，1：あり）
TxE：TxEMPTY
RRDY：RxRDY
TRDY：TxRDY</td></tr>
<tr><td>ライト</td><td>モード/コマンドライト
コマンドワード
<table><tr><th>D7</th><th>D6</th><th>D5</th><th>D4</th><th>D3</th><th>D2</th><th>D1</th><th>D0</th></tr><tr><td>EH</td><td>IR</td><td>RTS</td><td>ER</td><td>SBRK</td><td>RXE</td><td>DTR</td><td>TxEN</td></tr></table>EH：HUNTモードへ
IR：内部リセット（0：しない，1：する）
RTS：RTS信号
ER：エラーリセット（0：しない，1：する）
SBRK：センド・ブレイク・キャラクタ
RXE：受信許可/禁止（0：禁止，1：許可）
DTR：DTR信号
TxEN：送信許可/禁止（0：禁止，1：許可）</td></tr>
</table>

7．××××00××0011×A1A01　システムポート8255相当

<table>
<tr><th>I／O
アドレス</th><th>リード／
ライト</th><th>機　　　　能</th></tr>
<tr><td>31H</td><td>リード</td><td>ポートA（DIP　SW2）リード</td></tr>
<tr><td>33H</td><td>リード</td><td>ポートBリード
<table>
<tr><th>D7</th><th>D6</th><th>D5</th><th>D4</th><th>D3</th><th>D2</th><th>D1</th><th>D0</th></tr>
<tr><td>CI</td><td>CS</td><td>CD</td><td>INT3</td><td>CRTT</td><td>IMCK</td><td>EMCK</td><td>CDAT</td></tr>
</table>
CI：RS−232CのCI信号
CS：RS−232CのCS信号（送信可／不可）
CD：RS−232CのCD信号（キャリアの検出）
INT3：ハードディスク割り込み状態
CRTT：CRTのタイプ(0:15KHz,1:24KHz)
IMCK：内臓RAMのパリティエラー
EMCK：拡張RAMのパリティエラー
CDAT：カレンダ時計入力データ</td></tr>
<tr><td>35H</td><td>リード／ライト</td><td>ポートCリード／ライト
<table>
<tr><th>D7</th><th>D6</th><th>D5</th><th>D4</th><th>D3</th><th>D2</th><th>D1</th><th>D0</th></tr>
<tr><td>SH0</td><td>PSTM</td><td>SH1</td><td>MCKE</td><td>BUZ</td><td>TXRE</td><td>TXEE</td><td>RXRE</td></tr>
</table>
SH0：SHUT0（0：リセット後処理継続）
PSTM：プリンタのPSTB信号のマスク指定
SH1：SHUT1（0：リセットで処理中断）
MCKE：メモリチェック結果の格納（1：格納する）
BUZ：ブザー制御（0：鳴動，1：停止）
TXRE：RS−232C送信割り込み許可／禁止
TXEE：RS−232C TXEMPTY割り込み許可／禁止
RXRE：RS−232C受信割り込み許可／禁止</td></tr>
<tr><td>37H</td><td>ライト</td><td>8255モードライト
<table>
<tr><th>D7</th><th>D6</th><th>D5</th><th>D4</th><th>D3</th><th>D2</th><th>D1</th><th>D0</th></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
</table>
ポートC個別ライト
<table>
<tr><th>37Hへの出力値</th><th>動　　作</th></tr>
<tr><td>00H</td><td>RXRE　禁止</td></tr>
<tr><td>01H</td><td>RXRE　許可</td></tr>
<tr><td>02H</td><td>TXEE　禁止</td></tr>
<tr><td>03H</td><td>TXEE　許可</td></tr>
<tr><td>04H</td><td>TXRE　禁止</td></tr>
<tr><td>05H</td><td>TXRE　許可</td></tr>
<tr><td>06H</td><td>ブザー　ON</td></tr>
<tr><td>07H</td><td>ブザー　OFF</td></tr>
<tr><td>08H</td><td>MCKE　格納せず</td></tr>
<tr><td>09H</td><td>MCKE　格納</td></tr>
<tr><td>0AH</td><td>SHUT1←0</td></tr>
<tr><td>0BH</td><td>SHUT1←1</td></tr>
<tr><td>0CH</td><td>PSTB　マスクせず</td></tr>
<tr><td>0DH</td><td>PSTB　マスク</td></tr>
<tr><td>0EH</td><td>SHUT0←0</td></tr>
<tr><td>0FH</td><td>SHUT0←1</td></tr>
</table></td></tr>
</table>

8．××××00××0100×A1A00　プリンタインタフェース8255相当

<table>
<tr><th>I／O
アドレス</th><th>リード／
ライト</th><th>機　　　能</th></tr>
<tr><td>40H</td><td>ライト</td><td>ポートA（プリンタへの出力データ）ライト</td></tr>
<tr><td>42H</td><td>リード</td><td>ポートB（制御信号）リード
<table>
<tr><th>D7</th><th>D6</th><th>D5</th><th>D4</th><th>D3</th><th>D2</th><th>D1</th><th>D0</th></tr>
<tr><td>1</td><td>0</td><td>MOD</td><td>SW13</td><td>SW18</td><td>BSY</td><td>ACPU</td><td>0</td></tr>
</table>
MOD：システムックロックタイプ（0:10MHz系，1:8MHz系）
SW13：DIP　SW1－3の設定状態
SW18：DIP　SW1－8の設定状態
BSY：プリンタのBUSY信号（0：ビジー）
ACPU：動作中のCPU（1：Vシリーズ，0：その他）</td></tr>
<tr><td>44H</td><td>リード／ライト</td><td>ポートC（制御信号）リード／ライト
<table>
<tr><th>D7</th><th>D6</th><th>D5</th><th>D4</th><th>D3</th><th>D2</th><th>D1</th><th>D0</th></tr>
<tr><td>PSTB</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>
PSTB：プリンタのPSTB信号</td></tr>
<tr><td>46H</td><td>ライト</td><td>8255モードライト
<table>
<tr><th>D7</th><th>D6</th><th>D5</th><th>D4</th><th>D3</th><th>D2</th><th>D1</th><th>D0</th></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
</table>
ポートC個別ライト
<table>
<tr><th>46Hへの出力値</th><th>動　　作</th></tr>
<tr><td>0EH
0FH</td><td>PSTB　ON
PSTB　OFF</td></tr>
</table></td></tr>
</table>

9．××××00××0100××A01　キーボードインタフェース8251相当

<table>
<tr><th>I／O
アドレス</th><th>リード／
ライト</th><th>機　　　能</th></tr>
<tr><td>41H</td><td>リード／ライト</td><td>受信データリード／送信データライト</td></tr>
<tr><td rowspan="2">43H</td><td>リード</td><td>ステータスリード
<table>
<tr><th>D7</th><th>D6</th><th>D5</th><th>D4</th><th>D3</th><th>D2</th><th>D1</th><th>D0</th></tr>
<tr><td>DSR</td><td>×</td><td>FE</td><td>OE</td><td>PE</td><td>TxE</td><td>RRDY</td><td>TRDY</td></tr>
</table>
DSR：データセットレディ信号
FE：フレーミングエラー（0：なし，1：あり）
OE：オーバーランエラー（0：なし，1：あり）
PE：パリティエラー（0：なし，1：あり）
TxE：TxEMPTY
RRDY：RxRDY
TRDY：TxRDY</td></tr>
<tr><td>ライト</td><td>モード／コマンドライト
コマンドワード
<table>
<tr><th>D7</th><th>D6</th><th>D5</th><th>D4</th><th>D3</th><th>D2</th><th>D1</th><th>D0</th></tr>
<tr><td>×</td><td>IR</td><td>RTS</td><td>ER</td><td>SBRK</td><td>RXE</td><td>DTR</td><td>TxEN</td></tr>
</table>
IR：内部リセット（0：しない，1：する）
RTS：RTS信号
ER：エラーリセット（0：しない，1：する）
SBRK：センド・ブレイク・キャラクタ
RXE：受信許可／禁止（0：禁止，1：許可）
DTR：DTR信号
TxEN：送信許可／禁止（0：禁止，1：許可）</td></tr>
</table>

10．××××00××0101××A00　NMIコントロール

I／O アドレス	リード／ライト	機　　　能
50H	ライト	NMI　F／Fリセット（任意の値出力でNMI禁止）
52H	ライト	NMI　F／Fセット（任意の値出力でNMI許可）

11．××××00××0110A2A1A00　GDCμPD7220A相当（テキスト）

I／O アドレス	リード／ライト	機　　　能
60H	リード	GDCステータスリード D7: ×, D6: HBLN, D5: VSYN, D4: ×, D3: ×, D2: FEMP, D1: FFUL, D0: DTR HBLN：水平消去信号の発生を示す VSYN：垂直同期信号の発生を示す FEMP：FIFOが空であることを示す FFUL：FIFOがいっぱいであることを示す DTR：データが読み出し可能であることを示す
	ライト	GDCパラメータライト
62H	リード	GDCデータリード
	ライト	GDCコマンドライト
64H	ライト	CRT割り込みリセット（任意の値出力で直後のVSYNC発生時にCRTV割り込みがかかる）
68H	ライト	モードF／F1コントロール（下表参照）
		モードF／F2コントロール（下表参照）

モードF／F1コントロール

68Hへの出力値	動　作
00H	ATR4はバーチカルライン
01H	ATR4は簡易グラフ
02H	カラーグラフィックモード
03H	モノクログラフィックモード
04H	テキスト80字モード
05H	テキスト40字モード
06H	ANKは6×8ドット
07H	ANKは7×13ドット
08H	高解像度モード
09H	縦200ラインモード
0AH	コードアクセスモード
0BH	ビットマップモード
0CH	不揮発メモリ書き込み不可
0DH	不揮発メモリ書き込み可
0EH	画面表示不可
0FH	画面表示可

モードF／F2コントロール

6AHへの出力値	動　作
00H	8色グラフィックモード
01H	16色グラフィックモード
04H	GRCG互換モード*
05H	EGC拡張モード*

6AH　ライト

出力値	動作
06H	拡張モード変更不可
07H	拡張モード変更可
20H	標準グラフィックモード*
21H	拡張グラフィックモード*
26H	通常表示**
27H	全画面反転表示**
28H	通常重ね合わせ**
29H	演算付き重ね合わせ**
2AH	通常パレット書き込み**
2BH	高速パレット書き込み**
2CH	オーバースキャンカラーなし**
2DH	オーバースキャンカラーあり**
40H	CRTモード
41H	プラズマディスプレイモード
62H	プレーン形式**
63H	パックトピクセル形式**
68H	表ページ・裏ページは別個*
69H	表ページ・裏ページは連続*
6CH	上位ビットが左方向**
6DH	上位ビットが右方向**
84H	GDC　2．5MHzモード
83Hと85H	GDC　5MHzモード

* 拡張モード変更可のときのみ有効
** H98で拡張モード変更可のときのみ有効

6CH　リード*／ライト

ボーダーカラー

D7	D6	D5	D4	D3	D2	D1	D0
I*	G	R	B	0	0	0	0

I：ボーダーカラーインテンシティ
G：ボーダーカラー緑要素（0：なし，1：あり）
R：ボーダーカラー赤要素（0：なし，1：あり）
B：ボーダーカラー青要素（0：なし，1：あり）
* H98，MATEでのみ指定可能，リードはH98のみ

6EH　ライト

モードF／F3コントロール（H98のみ）

6EHへの出力値	動　　作
02H	拡張モード変更不可
03H	拡張モード変更可
08H	標準テキスト*
09H	1バイト拡張テキスト1*
0AH	1バイト拡張テキスト2*
0BH	1バイト拡張テキスト3*
0CH	拡大表示不可*
0DH	拡大表示可*
0EH	1バイトアトリビュート*
0FH	2バイト拡張アトリビュート*
14H	文字表示あり*
15H	文字表示なし*

* 拡張モード変更可のときのみ有効

12. ××××00××0111A2A1A00　CRTC・GRCG

I／O アドレス	リード／ライト	機　　　能
70H	リード／ライト*	キャラクタ位置ライン数ライト
72H	リード／ライト*	ボディフェイスライン数ライト
74H	リード／ライト*	キャラクタライン数ライト
76H	リード／ライト*	スムーススクロールライン数ライト
78H	リード／ライト*	スクロールエリア上辺位置行数ライト
7AH	リード／ライト*	スクロールエリア行数ライト
7CH	ライト	GRCGモードレジスタライト D7: CG　D6: RMW　D5: 0　D4: 0　D3: P3　D2: P2　D1: P1　D0: P0 CG：グラフィックチャージャの有効／無効（0:無効，1:有効） RMW：動作モード（0：ライト時TDWモード，リード時TCRモード 1：ライト時RMWモード，リード不可） P0～P3：プレーン0～3へのアクセス有効／無効 （0：有効，1：無効）
7EH	ライト	GRCGタイルレジスタライト

* リードはH98，MATEでのみ可能

13. ××××00××0111×A1A01 タイマコントローラ8253相当

I/O アドレス	リード/ライト	機能
71H	リード/ライト	カウンタ#0(タイマ割り込み)カウント値リード/ライト
73H	リード/ライト	カウンタ#1(ビープ)カウント値リード/ライト
75H	リード/ライト	カウンタ#2(通信速度)カウント値リード/ライト
77H	ライト	8253コントロールワードライト

8253コントロールワードライト

D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

SC1	SC0	
0	0	カウンタ#0
0	1	カウンタ#1
1	0	カウンタ#2

RL1	RL0	
0	0	カウント・ラッチ動作
0	1	LSBのリード/ライト
1	0	MSBのリード/ライト
1	1	LSB,MSBの順にリード/ライト

M2	M1	M0	
0	0	0	モード0
0	0	1	モード1
×	1	0	モード2
×	1	1	モード3
1	0	0	モード4
1	0	1	モード5

BCD	
0	バイナリ・カウント(2進16桁)
1	BCDカウント(10進4桁)

14. ××××××××100000A00 ハードディスクインタフェース

I/O アドレス	リード/ライト	機能
80H	リード/ライト	インタフェースデータバスリード/ライト
82H	リード	ステータスリード
	ライト	コントロール

15．XXXX00011 0001A1A00　FM音源YM2203(YM2608)

I/O アドレス	リード/ ライト	機　　　能
0188H	リード	YM2203/2608ステータスリード D7 D6 D5 D4 D3 D2 D1 D0 BUSY 0 0 0 0 0 FLGB FLGA BUSY：コマンド・データ送受信フラグ（0：送受信可） FLGB：タイマB　終了フラグ（1：カウント終了） FLGA：タイマA　終了フラグ（1：カウント終了）
	ライト	YM2203/2608内部アドレスライト
018AH	リード	YM2203/2608SSG音源部データリード
	ライト	YM2203/2608データライト
018CH	リード	YM2608ステータスリード D7 D6 D5 D4 D3 D2 D1 D0 BUSY 0 0 0 0 0 FLGB FLGA 各ビットの意味は0188Hのステータスと同じ
	ライト	YM2608拡張部内部アドレスライト
018EH	リード/ライト	YM2608拡張部データリード/ライト

16．XXXXXXXX10001A1A01　ネットワークインタフェースボード

I/O アドレス	リード/ ライト	機　　　能
89H	リード/ライト	RAM，インクリメントアドレスカウントリード/ライト
8BH	リード	INTリセット（リードによってリセットされる）
	ライト	アドレスカウントロウライト
8DH	リード	RAMリード
	ライト	ベクマストローブ
8FH	リード	ステータスリード
	ライト	アドレスカウンタハイライト

17．XXXXXXXX1001XA1A00　1MB　FDDコントローラ765相当

I/O アドレス	リード/ ライト	機　　　能
90H	リード	765ステータスレジスタリード
92H	リード/ライト	765データレジスタリード/ライト
94H	リード	リードスイッチ D7 D6 D5 D4 D3 D2 D1 D0 0 1 0 0 TYP1 TYP0 0 0 TYP1：FDD #3,#4のタイプ（0：1MB，1：両用） TYP0：FDD #1,#2のタイプ（0：1MB，1：両用）
	ライト	コントロールレジスタライト D7 D6 D5 D4 D3 D2 D1 D0 RST RDY 0 1 0 0 0 0 RST：765　LSIのリセット RDY：強制Ready

18．XXXXXXXX100110A01　GP－IBスイッチ

I／O アドレス	リード／ ライト	機　　　能
99H	リード	ボード上のSW1のリード（A0＝0）
9BH	リード	ボード上のSW1のリード（A0＝1）

19．XXXX00XX1010A2A1A00　GDCμPD7220A相当（グラフィック）

<table>
<tr><th>I／O
アドレス</th><th>リード／
ライト</th><th>機　　　能</th></tr>
<tr><td rowspan="2">A0H</td><td>リード</td><td>GDCステータスリード
<table><tr><td>D7</td><td>D6</td><td>D5</td><td>D4</td><td>D3</td><td>D2</td><td>D1</td><td>D0</td></tr><tr><td>×</td><td>HBLN</td><td>VSYN</td><td>×</td><td>DRAW</td><td>FEMP</td><td>FFUL</td><td>DTR</td></tr></table>HBLN：水平消去信号の発生を示す
VSYN：垂直同期信号の発生を示す
DRAW：描画中であることを示す
FEMP：FIFOが空であることを示す
FFUL：FIFOがいっぱいであることを示す
DTR：データが読み出し可能であることを示す</td></tr>
<tr><td>ライト</td><td>GDCパラメータライト</td></tr>
<tr><td rowspan="2">A2H</td><td>リード</td><td>GDCデータリード</td></tr>
<tr><td>ライト</td><td>GDCコマンドライト</td></tr>
<tr><td>A4H</td><td>リード／ライト *</td><td>表示ページ
<table><tr><td>D7</td><td>D6</td><td>D5</td><td>D4</td><td>D3</td><td>D2</td><td>D1</td><td>D0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>DP</td></tr></table>DP：表示ページ（0：表画面，1：裏画面）</td></tr>
<tr><td>A6H</td><td>リード／ライト *</td><td>描画ページ
<table><tr><td>D7</td><td>D6</td><td>D5</td><td>D4</td><td>D3</td><td>D2</td><td>D1</td><td>D0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>WP</td></tr></table>WP：描画ページ（0：表画面，1：裏画面）</td></tr>
<tr><td>A8H</td><td>リード*／
ライト</td><td>パレット番号（16色／256色モード）
パレット＃3，＃7色要素（8色モード）</td></tr>
<tr><td>AAH</td><td>リード*／
ライト</td><td>パレット設定色緑輝度（16色／256色モード）
パレット＃1，＃5色要素（8色モード）</td></tr>
<tr><td>ACH</td><td>リード*／
ライト</td><td>パレット設定色赤輝度（16色／256色モード）
パレット＃2，＃6色要素（8色モード）</td></tr>
<tr><td>AEH</td><td>リード*／
ライト</td><td>パレット設定色青輝度（16色／256色モード）
パレット＃0，＃4色要素（8色モード）</td></tr>
</table>

＊ リードはH98およびMATEでのみ可能

20．×××××01××1010A2A1A00　EGC制御

I／O アドレス	リード／ライト	機能
04A0H	リード／ライト*	描画／比較アクセスプレーン
04A2H	リード／ライト*	パターンデータの種類・リードプレーン番号
04A4H	リード／ライト*	描画／リード／パターンレジスタアクセスモード
04A6H	リード／ライト*	フォアグラウンドカラー
04A8H	リード／ライト*	マスクレジスタ
04AAH	リード／ライト*	バックグラウンドカラー
04ACH	リード／ライト*	ビットアドレス・ブロック転送方向
04AEH	リード／ライト*	ライト領域のビット長

*リードはE²GCでのみ可能．ライトはEGC拡張モードかつグラフィックチャージャ有効のときのみ可能

21．××××10011010A2A1A00　グラフィック制御（H98,MATEのみ）

I／O アドレス	リード／ライト	機能
09A0H	リード	モードF／F2等リード

D7	D6	D5	D4	D3	D2	D1	D0
×	×	×	×	×	×	GC*	FF

GC：GDCの動作クロック(0:2.5MHz､1:5MHz)
FF：モードF／F2の設定状態

09A0Hへの出力値	FF	現在の設定状態
04H	0	8色グラフィックモード
	1	16色グラフィックモード
05H	0	CRTモード
	1	プラズマディスプレイモード
07H	0	GRCG互換モード
	1	EGC拡張モード
08H	0	拡張モード変更不可
	1	拡張モード変更可
0AH	0	標準グラフィックモード
	1	拡張グラフィックモード
0BH	0	プレーン形式*
	1	パックトピクセル形式*
0DH	0	表ページ・裏ページは別個
	1	表ページ・裏ページは連続
11H	0	通常表示*
	1	全画面反転表示*
12H	0	オーバースキャンカラーなし*
	1	オーバースキャンカラーあり*
13H	0	通常重ね合わせ*
	1	演算付き重ね合わせ*
17H	0	上位ビットが左方向*
	1	上位ビットが右方向*
18H	0	通常パレット書き込み*
	1	高速パレット書き込み*

		＊Ｈ９８のみ
	ライト	リードするモードＦ／Ｆ２番号ライト
０９Ａ２Ｈ	リード	パレットステータスリード（Ｈ９８のみ） D7 D6 D5 D4 D3 D2 D1 D0 × × × × × × × ST ＳＴ：２５６色通常パレット書き込み時のビジーフラグ （０：パレット書き込み可，１：ビジー）
０９Ａ６Ｈ	リード／ライト	オーバースキャンカラーリード／ライト（Ｈ９８のみ）
０９Ａ８Ｈ	リード／ライト	ＣＲＴ水平スキャン周波数リード／ライト D7 D6 D5 D4 D3 D2 D1 D0 0 0 0 0 0 0 0 HF ＨＦ：水平スキャン周波数(0:24KHz、1:31KHz)
０９ＡＥＨ	リード／ライト	プレーンマスクリード／ライト（Ｈ９８のみ） D7 D6 D5 D4 D3 D2 D1 D0 M7 M6 M5 M4 M3 M2 M1 M0 M0〜M7：プレーンマスク（０：マスク，１：表示）

２２．××××００××１０１０A2A1A0１　ＫＣＧ　ＲＯＭ

Ｉ／Ｏ アドレス	リード／ ライト	機　　能
Ａ１Ｈ	リード／ライト*	文字コード第２バイト
Ａ３Ｈ	リード／ライト*	文字コード第１バイト
Ａ５Ｈ	リード／ライト*	文字パターン行指定カウンタ D7 D6 D5 D4 D3 D2 D1 D0 0 0 L/$\overline{R}$ RC4 RC3 RC2 RC1 RC0 RC3 RC2 RC1 RC0　L/$\overline{R}$=1 (D7 D6 D5 D4 D3 D2 D1 D0)　L/$\overline{R}$=0 (D7 D6 D5 D4 D3 D2 D1 D0) 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 1 0 1 0 0 0 1 0 1 0 1 1 0 0 1 1 1 1 0 0 0 1 0 0 1 1 0 1 0 1 0 1 1 1 1 0 0 1 1 0 1 1 1 1 0 1 1 1 1
Ａ９Ｈ	リード／ライト	文字パターンリード／ライト

＊リードはＨ９８，ＭＡＴＥでのみ可能

２３．××××××××１０１１A3A2A1A0　通信制御アダプタ

<table>
<tr><th>I／O
アドレス</th><th>リード／
ライト</th><th>機　　　　能</th></tr>
<tr><td>B０H</td><td>リード</td><td>D７２０１チャネルAデータリード</td></tr>
<tr><td>B２H</td><td>ライト</td><td>D７２０１チャネルBデータライト</td></tr>
<tr><td rowspan="2">B４H</td><td>リード</td><td>D７２０１チャネルAステータスリード</td></tr>
<tr><td>ライト</td><td>D７２０１チャネルAコマンドライト</td></tr>
<tr><td rowspan="2">B６H</td><td>リード</td><td>D７２０１チャネルBステータスリード</td></tr>
<tr><td>ライト</td><td>D７２０１チャネルBコマンドライト</td></tr>
<tr><td>B１H</td><td>リード／ライト</td><td>８２５５ポートAデータリード／ライト</td></tr>
<tr><td>B３H</td><td>リード／ライト</td><td>８２５５ポートB（DIP　SW）データリード</td></tr>
<tr><td>B５H</td><td>リード／ライト</td><td>８２５５ポートCデータリード／ライト</td></tr>
<tr><td>B９H</td><td>リード／ライト</td><td>８２５３カウンタ＃０カウント値リード／ライト</td></tr>
<tr><td>BBH</td><td>リード／ライト</td><td>８２５３カウンタ＃１カウント値リード／ライト</td></tr>
<tr><td>BDH</td><td>リード／ライト</td><td>８２５３カウンタ＃２カウント値リード／ライト</td></tr>
<tr><td>BFH</td><td>ライト</td><td>８２５３モードライト
<table><tr><td>D7</td><td>D6</td><td>D5</td><td>D4</td><td>D3</td><td>D2</td><td>D1</td><td>D0</td></tr><tr><td>SC1</td><td>SC0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td></tr></table></td></tr>
</table>

２４．××××××××１０１１A3A2A1A0　RS-232C拡張インタフェース

<table>
<tr><th>I／O
アドレス</th><th>リード／
ライト</th><th>機　　　　能</th></tr>
<tr><td rowspan="2">B０H</td><td>リード</td><td>チャネル２シグナル・スイッチリード
<table><tr><td>D7</td><td>D6</td><td>D5</td><td>D4</td><td>D3</td><td>D2</td><td>D1</td><td>D0</td></tr><tr><td>$\overline{CI}$</td><td>$\overline{CS}$</td><td>$\overline{CD}$</td><td>×</td><td>×</td><td>×</td><td>IR1</td><td>IR2</td></tr></table></td></tr>
<tr><td>ライト</td><td>チャネル２マスクセット
<table><tr><td>D7</td><td>D6</td><td>D5</td><td>D4</td><td>D3</td><td>D2</td><td>D1</td><td>D0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>TXR</td><td>TXE</td><td>RXR</td></tr></table></td></tr>
<tr><td rowspan="2">B２H</td><td>リード</td><td>チャネル３シグナル・スイッチリード</td></tr>
<tr><td>ライト</td><td>チャネル３マスクセット</td></tr>
<tr><td>B１H</td><td>リード／ライト</td><td>チャネル２データリード／ライト</td></tr>
<tr><td>B３H</td><td>リード</td><td>チャネル２ステータスリード</td></tr>
<tr><td>B９H</td><td>リード／ライト</td><td>チャネル３データリード／ライト</td></tr>
<tr><td rowspan="2">BBH</td><td>リード</td><td>チャネル３ステータスリード</td></tr>
<tr><td>ライト</td><td>チャネル３モード・コマンドセット</td></tr>
</table>

２５．XXXXXXXX１０１１１１１０　1MB/640KB　FDD切換えインタフェース

Ｉ／Ｏ アドレス	リード／ライト	機　　能
ＢＥＨ	リード	リードモードステータス D7: × / D6: × / D5: × / D4: × / D3: SW32 / D2: SW31 / D1: FDE / D0: PE ＳＷ３２：ＤＩＰ　ＳＷ３－２の設定状態 ＳＷ３１：ＤＩＰ　ＳＷ３－１の設定状態 ＦＤＥ：ＦＤＤ切りかえ状況(0:640KB、1:1MB) ＰＥ：Ｉ／Ｏポート切りかえ状況(0:640KB、1:1MB)
	ライト	ライトモードチェンジ D7: 0 / D6: 0 / D5: 0 / D4: 0 / D3: 0 / D2: 0 / D1: FDE / D0: PE ＦＤＥ：ＦＤＤ切りかえ(0:640KB、1:1MB) ＰＥ：Ｉ／Ｏポート切りかえ(0:640KB、1:1MB)

２６．XXXXXXXX１１００１A1A0０　６４０ＫＢ　ＦＤＤコントローラ765相当

Ｉ／Ｏ アドレス	リード／ライト	機　　能
Ｃ８Ｈ	リード	７６５ステータスレジスタリード
ＣＡＨ	リード／ライト	７６５データレジスタリード／ライト
ＣＣＨ	リード	リードスイッチ D7: 0 / D6: 1 / D5: 1 / D4: RDY / D3: TYP1 / D2: TYP0 / D1: 0 / D0: 0 RDY：ＦＤＤの状態信号（０：ノットレディ，１：レディ） TYP1：FDD #3,#4のタイプ（０：１ＭＢ，１：両用） TYP0：FDD #1,#2のタイプ（０：１ＭＢ，１：両用）
	ライト	コントロールレジスタライト D7: RST / D6: EAI1 / D5: EAI0 / D4: DMAE / D3: MTON / D2: TMSK / D1: × / D0: TTRG ＲＳＴ：７６５　ＬＳＩのリセット EAI1=0, EAI0=1：アテンション割り込み禁止 EAI1=1, EAI0=0：アテンション割り込み許可 ＤＭＡＥ：ＤＭＡ許可（０：禁止，１：許可） ＭＴＯＮ：モータ制御 ＴＭＳＫ：モータ制御タイマの割り込み要求マスク （０：割り込みマスク，１：割り込み可） ＴＴＲＧ：モータ制御タイマの起動

２７．××××××××１１００A2A1A0１　ＧＰ－ＩB7210相当

Ｉ／Ｏ アドレス	リード／ ライト	機　　　　　能
Ｃ１Ｈ	リード	データ　イン
	ライト	バイト　アウト
Ｃ３Ｈ	リード	インタラプト　ステータス１
	ライト	インタラプト　マスタ　１
Ｃ５Ｈ	リード	インタラプト　ステータス２
	ライト	インタラプト　マスタ　２
Ｃ７Ｈ	リード	シリアル・ポール　ステータス
	ライト	シリアル・ポート　モード
Ｃ９Ｈ	リード	アドレス　ステータス
	ライト	アドレス　モード
ＣＢＨ	リード	コマンド　パス　スルー
	ライト	オグジッリアリ　モード
ＣＤＨ	リード	アドレス　０
	ライト	アドレス　０／１
ＣＦＨ	リード	アドレス　１
	ライト	エンド　オブ　ストリング

28. 011111111011A1A01　マウスコントローラ8255相当

<table>
<tr><th>I／O
アドレス</th><th>リード／
ライト</th><th>機　　　能</th></tr>
<tr><td>7FD9H</td><td>リード</td><td>8255ポートA（マウスデータ）リード
<table>
<tr><td>D7</td><td>D6</td><td>D5</td><td>D4</td><td>D3</td><td>D2</td><td>D1</td><td>D0</td></tr>
<tr><td>LEFT</td><td>×</td><td>RIGH</td><td>×</td><td>MD3</td><td>MD2</td><td>MD1</td><td>MD0</td></tr>
</table>
LEFT：マウス左ボタンの状態（0：押されている，1：放されている）
RIGH：マウス右ボタンの状態（0：押されている，1：放されている）
MD0～3：SXY，SHLで選択されたマウス差動カウント値の上位あるいは下位4ビット</td></tr>
<tr><td>7FDBH</td><td>リード</td><td>8255ポートBリード
Bit1：クロックスイッチの状態（286機のみ）
Bit6：DIP　SW3－6の設定状態</td></tr>
<tr><td rowspan="2">7FDDH</td><td>リード</td><td>8255ポートCリード
<table>
<tr><td>D7</td><td>D6</td><td>D5</td><td>D4</td><td>D3</td><td>D2</td><td>D1</td><td>D0</td></tr>
<tr><td>HC</td><td>SXY</td><td>SHL</td><td>INT</td><td>MMOD</td><td>SW38</td><td>SW16</td><td>SW15</td></tr>
</table>
HC，SXY，SHL，INT：ライト部参照
MMOD：マシンの状態（0：ノーマル専用機，
1：ノーマル・ハイレゾ両用機のノーマル）
SW38：DIP　SW3－8の設定状態
SW16：DIP　SW1－6の設定状態
SW15：DIP　SW1－5の設定状態</td></tr>
<tr><td>ライト</td><td>8255ポートCライト
<table>
<tr><td>D7</td><td>D6</td><td>D5</td><td>D4</td><td>D3</td><td>D2</td><td>D1</td><td>D0</td></tr>
<tr><td>HC</td><td>SXY</td><td>SHL</td><td>INT</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>
HC：0→1でリードバッファにカウント値がコピーされ，カウンタがクリアされる
SXY：XあるいはY方向カウント値読み出し指定
（0：X方向，1：Y方向）
SHL：上位あるいは下位4ビット読み出し指定
（0：下位4ビット，1：上位4ビット）
INT：マウス割り込みのマスク（0：許可，1：マスク）</td></tr>
<tr><td>7FDFH</td><td>ライト</td><td>8255モードライト
<table>
<tr><td>D7</td><td>D6</td><td>D5</td><td>D4</td><td>D3</td><td>D2</td><td>D1</td><td>D0</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
</table>
ポートC個別ライト
<table>
<tr><th>7FDFHへの出力値</th><th>動　　作</th></tr>
<tr><td>08H</td><td>マウス割り込み許可</td></tr>
<tr><td>09H</td><td>マウス割り込み禁止</td></tr>
<tr><td>0EH</td><td>カウント値読み出し準備</td></tr>
<tr><td>0FH</td><td>カウント値読み出し</td></tr>
</table></td></tr>
</table>

29. 0011111111011A011 タイマコントローラ8253相当（ビープ音制御）

I／O アドレス	リード／ライト	機 能
3FDBH	リード／ライト	カウンタ＃1（ビープ）カウント値リード／ライト
3FDFH	ライト	8253コントロールワードライト D7 D6 D5 D4 D3 D2 D1 D0 0 1 1 1 0 1 1 0

30. 1011111111011011 マウス割り込み時間間隔設定

I／O アドレス	リード／ライト	機 能
BFDBH	リード／ライト*	マウス割り込み時間間隔

BFDBH への出力値	時間間隔
00H	8ms
01H	16ms
02H	34ms
03H	67ms

* リードはH98，MATEでのみ可能，リード時には上位6ビットはすべて1になる

31. ××××00××11110A1A00 CPU

I／O アドレス	リード／ライト	機 能
F0H	ライト	CPUリセット D7 D6 D5 D4 D3 D2 D1 D0 0 0 0 0 0 0 0 0 リセット後の動作はシステムポートの設定値によって異なる
F2H	ライト	アドレスバスA20マスク解除 D7 D6 D5 D4 D3 D2 D1 D0 0 0 0 0 0 0 0 0

§4-4 I/Oアクセス時の必要ウェイト数一覧

80286以降のCPUはパイプライン処理を行なっているので，同じ命令を同じCPUで実行しても処理時間が著しく異なることがあり得ます．そのため，I/O制御などのタイミングが重要な処理に関しては，このことを考慮に入れてプログラムを組む必要があります．パイプライン処理による命令の実行速度の変化を防ぐには，JMP $+2命令などでパイプラインをクリアしてやればよいでしょう．ここでは，それぞれのCPUの場合に，どのデバイスにアクセスするとき，どのくらいウェイトを取ればよいかを表で示します．

■80286CPU［JMP　$＋2］挿入数

周辺ＬＳＩ	OUT→OUT			IN→IN			OUT→IN			IN→OUT		
	8	10	12	8	10	12	8	10	12	8	10	12(MHz)
8237 ＤＭＡＣ	1	1	1	0	0	0	1	1	1	0	0	0
8253 タイマ	1	1(2)	2	1	1	2	1	1(2)	2	1	1	2
8255 ＰＩＯ	1	1(2)	1	1	1	1	1	1(2)	1	1	1	1
8259 ＰＩＣ	1	0(1)	0	0	0	0	1	1	1	0	0	0
8251 モード初期化	3	3	3	-	-	-	-	-	-	-	-	-
SIO ライトデータ同期	4	4	4	-	-	-	-	-	-	-	-	-
ライトデータ非同期	7(6)	7	7	-	-	-	-	-	-	-	-	-
その他	-	-	-	0	0	0	0	0	0	0	0	0
765 ＦＤＣ	0	0	0	0	0	0	0	0	0	0	0	0
7220 ノーマル　標準CRT	2	3	3	2	2	2	2	3	3	2	2	2
GDC グラフ2.5M　高解像CRT	2	2	2	1	1	1	2	2	2	1	1	1
ノーマルグラフ 標準CRT	1	1	1	1	1	1	1	1	2	1	1	1
5Mテキスト　高解像CRT	1	1	1	0	0	0	1	1	1	0	0	0
ハイレゾ　グラフ	1	1	-	1	1	-	1	1	-	1	1	-
ＸＡ／ＸＬ　テキスト	1	-	1*	1	-	1	1	-	1*	1	-	
7201 通信制御	1	1	1	0	0	0	1	1	1	0	0	0
7210 ＧＢ－ＩＢ	1*	1	1	0	0	0	1*	1	1	0	0	0

()内はＸＡ／ＸＬ　＊ＸＡでは0

■80386（SX）CPU［JMP　$＋2］挿入数

周辺ＬＳＩ	OUT→OUT			IN→IN			OUT→IN			IN→OUT		
	12	16	20	12	16	20	12	16	20	12	16	20(MHz)
8237 ＤＭＡＣ	1	1	1	0	0	0	1	1	1	0	0	0
8253 タイマ	2	2	2	1	1	2	2	2	2	1	1	2
8255 ＰＩＯ	2	2	2	1	1	1	2	2	2	1	1	1
8259 ＰＩＣ	0	0	0	0	0	0	1	1	1	0	0	0
8251 モード初期化	6	6	6	-	-	-	-	-	-	-	-	-
SIO ライトデータ同期	8	8	8	-	-	-	-	-	-	-	-	-
ライトデータ非同期	16	16	16	-	-	-	-	-	-	-	-	-
その他	-	-	-	0	0	0	0	0	0	0	0	0
765 ＦＤＣ	0	0	0	0	0	0	0	0	0	0	0	0
7220 ノーマルグラフ2.5M	3	3	3	2	2	3	3	3	3	2	2	3
GDC ノーマルグラフ5M	1	1	1	0	0*1	0*1	1	1	1	0	0*1	0*1
ノーマルテキスト	1	1	1	2	0*2	0*3	1	1	1	0	0*4	0*3
ハイレゾグラフ	-	1	1	-	0	0	-	1	1	-	0	0
ハイレゾテキスト	-	2	2	-	1	2	-	2	2	-	1	2
7201 通信制御	1	1	1	0	0	0	1	1	1	0	0	0
7210 ＧＢ－ＩＢ	1	1	1	0	0	0	1	1	1	0	0	0

*1 PC-9801DAでは1
*2 PC-9801DAでは1　PC-9801RS21，51では2　PC-9801NS/Eでは2
*3 PC-9801DAでは1　PC-9801Tシリーズでは2
*4 PC-9801DAでは1　PC-9801RS21，51では2

§4-5 割り込みベクター覧表

ベクタ番号	用途
0 0	除算エラー
0 1	シングルステップ
0 2	NMI
0 3	INT 3
0 4	オーバーフロー
0 5	COPYキー
0 6	STOPキー
0 7	インターバルタイマ
0 8	タイマ
0 9	キーボード
0 A	V－SYNC
0 B	拡張バスINT 0
0 C	RS－232C
0 D	拡張バスINT 1
0 E	拡張バスINT 2（ハイレゾモード時，マウスで利用）
0 F	予約
1 0	NDP
1 1	拡張バスINT 3（主に，HDDで利用）
1 2	拡張バスINT 4 1（640k　FDDで利用）
1 3	拡張バスINT 4 2（1M　FDDで利用）
1 4	拡張バスINT 5（主にFM音源で利用）
1 5	拡張バスINT 6（ノーマルモード時，マウスで利用）
1 6	プリンタ（ハイレゾモードのみ）
1 7	予約
1 8	キーボード／CRT　BIOS
1 9	RS－232C　BIOS
1 A	プリンタBIOS
1 B	DISK　BIOS
1 C	カレンダBIOS
1 D	予約
1 E	BASIC
1 F	予約
2 0－3 F	予約（DOSのファンクションコールなどで利用）
4 0－7 F	ユーザー用（EMSファンクションなどで利用）

§4-6 各命令の所要クロック数一覧

■ＣＰＵ命令

●表の読み方

◆命令

rel8	命令の最後から前方へ128バイトから命令の最後から後方へ127バイトまでの相対アドレス
rel16,rel32	アセンブルされる命令と同一コードセグメント内の相対アドレス
ptr16:16,ptr16:32	通常は命令のコードセグメントとは異なるコードセグメントうちのfarポインタ
r8	バイトレジスタAL，CL，DL，BL，AH，CH，DH，BHのうちの１つ
r16	ワードレジスタAX，CX，DX，BX，SP，BP，SI，DIのうちの１つ
r32	ダブルワードレジスタEAX，ECX，EDX，EBX，ESP，EBP，ESI，EDIの内の１つ
imm8,imm16,imm32	即値バイトの値
r/m8	バイトレジスタまたはメモリ上の内容
r/m16	ワードレジスタまたはメモリ上の内容
r/m32	ダブルワードレジスタまたはメモリ上の内容
m8	DS：SIまたはES：DIがアドレスするメモリバイト
m16	DS：SIまたはES：DIがアドレスするメモリワード
m32	DS：SIまたはES：DIがアドレスするメモリダブルワード
m16:16,m16:32	２つの数からなるfarポインタを含むメモリオペランド
m16 & 32, m16&16, m32 & 32	アンパサンドの左右にサイズが示される，データ項目の対からなるメモリオペランド
moffs8,moffs16, moffs32	BYTE，WORDまたはDWORD型のメモリ変数
Sreg	セグメントレジスタ

◆“４８６”，“３８６”，“２８６”

命令を実行するのに必要なクロックサイクル数を示します．

r/mオペランドをもつ命令のクロック数は，スラッシュで区切られています．左側がレジスタオペランドのときで，右側がメモリオペランドのときです．

クロック数の指定の記号

n	繰り返しの回数
m	次に実行される命令を構成する要素数
ｐｍ＝	保護モードで実行するときに適用されるクロック数

命令	486	386	286
AAA	3	4	3
AAD	14	19	14
AAM	15	16	83
AAS	3	4	3
ADC r/m8,r8	1/3	2/7	2/7
ADC r/m16,r16	1/3	2/7	2/7
ADC r/m32,r32	1/3	2/7	
ADC r8,r/m8	1/2	2/6	2/7
ADC r16,r/m16	1/2	2/6	2/7
ADC r32,r/m32	1/2	2/6	
ADC AL,imm8	1	2	3
ADC AX,imm16	1	2	3
ADC EAX,imm32	1	2	
ADC r/m8,imm8	1/3	2/7	3/7
ADC r/m16,imm16	1/3	2/7	3/7
ADC r/m32,imm32	1/3	2/7	
ADC r/m16,imm8	1/3	2/7	3/7
ADC r/m32,imm8	1/3	2/7	
ADD AL,imm8	1	2	3
ADD AX,imm16	1	2	3
ADD EAX,imm32	1	2	
ADD r/m8,imm8	1	2/7	3/7
ADD r/m16,imm16	1/3	2/7	3/7
ADD r/m32,imm32	1/3	2/7	
ADD r/m16,imm8	1/3	2/7	3/7
ADD r/m32,imm8	1/3	2/7	
ADD r/m8.r8	1/3	2/7	2/7
ADD r/m16,r16	1/3	2/7	2/7
ADD r/m32,r32	1/3	2/7	
ADD r8,r/m8	1/2	2/6	2/7
ADD r16,r/m16	1/2	2/6	2/7
ADD r32,r/m32	1/2	2/6	
AND r/m8,r8	1/3	2/7	2/7
AND r/m16,r16	1/3	2/7	2/7
AND r/m32,r32	1/3	2/7	
AND r8,r/m8	1/2	2/6	2/7
AND r16,r/m16	1/2	2/6	2/7
AND r32,r/m32	1/2	2/6	
AND AL,imm8	1	2	3
AND AX,imm16	1	2	3

命令	486	386	286
ＡＮＤ　EAX,imm32	1	2	
ＡＮＤ　r/m8,imm8	1/3	2/7	3/7
ＡＮＤ　r/m16,imm16	1/3	2/7	3/7
ＡＮＤ　r/m32.imm32	1/3	2/7	
ＡＮＤ　r/m16,imm8	1/3	2/7	
ＡＮＤ　r/m32,imm8	1/3	2/7	
ＡＲＰＬ　r/m16,r16	9/9	pm=20/21	pm=10/11
ＢＯＵＮＤ　r16,m16&16	7	10	13
ＢＯＵＮＤ　r32,m32&32	7	10	
ＢＳＦ　r16,r/m16	6〜43/7〜43	10+3n	
ＢＳＦ　r32,r/m32		10+3n	
ＢＳＲ　r16,r/m16	6〜103/7〜104	10+3n	
ＢＳＲ　r32,r/m32	6〜103/7〜104	10+3n	
ＢＳＷＡＰ　r32	1		
ＢＴ　r/m16,r16	3/8	3/12	
ＢＴ　r/m32,r32	3/8	3/12	
ＢＴ　r/m16,imm8	3/3	3/6	
ＢＴ　r/m32,imm8	3/3	3/6	
ＢＴＣ　r/m16,r16	6/13	6/13	
ＢＴＣ　r/m32,r32	6/13	6/13	
ＢＴＣ　r/m16,imm8	6/8	6/8	
ＢＴＣ　r/m32,imm8	6/8	6/8	
ＢＴＲ　r/m16,r16	6/13	6/13	
ＢＴＲ　r/m32,r32	6/13	6/13	
ＢＴＲ　r/m16,imm8	6/8	6/8	
ＢＴＲ　r/m32,imm8	6/8	6/8	
ＢＳＴ　r/m16,r16	6/13	6/13	
ＢＳＴ　r/m32,r32	6/13	6/13	
ＢＳＴ　r/m16,imm8	6/8	6/8	
ＢＳＴ　r/m32.imm8	6/8	6/8	
ＣＡＬＬ　rel16	3	7+m	7
ＣＡＬＬ　r/m16	5/5	7+m/10+m	7/11
ＣＡＬＬ　ptr16:16	18,pm=20	17+m,pm=34+m	13,pm=26
ＣＡＬＬ　ptr16:16	pm=35	pm=52+m	41
ＣＡＬＬ　ptr16:16	pm=69	pm=86+m	82
ＣＡＬＬ　ptr16:16	pm=77+4x	pm=94+4x+m	86+4x
ＣＡＬＬ　ptr16:16	pm=37+ts	ts	177/182
ＣＡＬＬ　m16:16	17,pm=20	22+m,pm=38+m	16/29
ＣＡＬＬ　m16:16	pm=35	pm=56+m	44
ＣＡＬＬ　m16:16	pm=69	pm=90+m	83
ＣＡＬＬ　m16:16	pm=77+4x	pm=98+4x+m	90+4x

命令	486	386	286
CALL m16:16	pm=37+ts	5+ts	180/185
CALL rel32	3	7+m	
CALL r/m32	5/5	7+m/10+m	
CALL ptr16:32	18,=20	17+m,pm=34+m	
CALL ptr16:32	pm=35	pm=52+m	
CALL ptr16:32	pm=69	pm=84+m	
CALL ptr32:32	pm=77+4x	pm=94+4x+m	
CALL ptr16:32	pm=37+ts	ts	
CALL ptr16:32	17,pm=20	22+m,pm=38+m	
CALL ptr16:32	pm=35	pm=56+m	
CALL ptr16:32	pm=69	pm=90+m	
CALL ptr16:32	pm=77+4x	pm=98+4x+m	
CALL ptr16:32	pm=37+ts	5+ts	
CBW	3	3	2
CDQ	3	2	
CLC	2	2	2
CLD	2	2	2
CLI	5	3	3
CLTS	7	5	2
CMC	2	2	2
CMP AL,imm8	1	2	3
CMP AX,imm16	1	2	3
CMP EAX,imm32	1	2	
CMP r/m8,imm8	1/2	2/5	3/6
CMP r/m16,imm16	1/2	2/5	3/6
CMP r/m32,imm32	1/2	2/5	
CMP r/m16.imm8	1/2	2/5	3/6
CMP r/m32,imm8	1/2	2/5	
CMP r/m8,r8	1/2	2/5	2/7
CMP r/m16,r16	1/2	2/5	2/7
CMP r/m32,r32	1/2	2/5	
CMP r8,r/m8	1/2	2/5	2/6
CMP r16,r/m16	1/2	2/6	2/6
CMP r32,r/m32	1/2	2/6	
CMPS m8,m8	8	10	8
CMPS m16,m16	8	10	8
CMPS m32,m32	8	10	
CMPSB	8	10	8
CMPSW	8	10	8
CMPSD	8	10	
CMPXCHG r/m8,r8 *	6/7;6/10		

命令	486	386	286
CMPXCHG　r/m16,r16　*	6/7;6/10		
CMPXCHG　r/m32,r32　*	6/7;6/10		
CWD	3	2	
CWDE	3	3	
DAA	2	4	3
DAS	2	4	3
DEC　r/m8	1/3	2/6	2/7
DEC　r/m16	1/3	2/6	2/7
DEC　r/m32	1/3	2/6	
DEC　r16	1	2	2
DEC　r32	1	2	
DIV　AL,r/m8	16/16	14/17	
DIV　AX,r/m16	24/24	22/25	
DIV　EAX,r/m32	40/40	38/41	
Enter　imm16,0	14	10	11
Enter　imm16,1	17	12	15
Enter　imm16,imm18	17+3n	15+4(n〜1)	15+4(n〜1)
HLT	4	5	2
IDIV　r/m8	19/20	19	17/20
IDIV　AX,imm16	27/28	27	25/28
IDIV　EAX,r/m32	43/44	43	
IMUL　r/m8	13〜18/13〜18	9〜14/12〜17	13/16
IMUL　r/m16	13〜26/13〜26	9〜22/12〜25	21/24
IMUL　r/m32	12〜42/13〜42	9〜38/12〜41	
IMUL　r16,r/m16	13〜42/13〜42	9〜22/12〜25	
IMUL　r32,r/m32	13〜26/13〜26	9〜38/12〜41	
IMUL　r16,r/m16,imm8	13〜26/13〜26	9〜14/12〜17	21/24
IMUL　r32,r/m32,imm8	13〜42	9〜14/12〜17	
IMUL　r16,imm8	13〜26	9〜14/12〜17	21/24
IMUL　r32,imm8	13〜42	9〜14/12〜17	
IMUL　r16,r/m16,imm16	13〜26/13〜26	9〜22/12〜25	21/24
IMUL　r32,r/m32,imm32	13〜42/13〜42	9〜38/12〜41	
IMUL　r16,imm16	13〜26/13〜26	9〜22/12〜25	
IMUL　r32,imm32	13〜42/13〜42	9〜38/12〜41	
IN　AL,imm8　*	14,pm=8/28,vm=27	12,pm=6/26	5
IN　AX,imm8　*	14,pm=8/28,vm=27	12,pm=6/26	5
IN　EAX,imm8　*	14.pm=8/28,vm=27	12,pm=6/26	
IN　AL,DX　*	14,pm=8/28,vm=27	13,pm=7/27	5
IN　AX,DX　*	14,pm=8/28,vm=27	13,pm=7/27	5
IN　EAX,DX　*	14,pm=8/28,vm=27	13,pm=7/27	
INC　r/m8	1/3	2/6	2/7

命令	486	386	286
ＩＮＣ　r/m16	1/3	2/6	2/7
ＩＮＣ　r/m32	1/3		
ＩＮＣ　r16	1	2	2
ＩＮＣ　r32	1		
ＩＮＳ　r/m8,DX　*	17,pm=10/32, vm=30	15,pm=9/29	5
ＩＮＳ　r/m16,DX　*	17,pm=10/32, vm=30	15,pm=9/29	5
ＩＮＳ　r/m32,DX　*	17,pm=10/32, vm=30	15,pm=9/29	
ＩＮＳＢ　*	17,pm=10/32, vm=30	15,pm=9/29	5
ＩＮＳＷ　*	17,pm=10/32, vm=30	15,pm=9/29	5
ＩＮＳＤ　*	17,pm=10/32, vm=30	15,pm=9/29	
ＩＮＴ３	26	33	23
ＩＮＴ３	44	pm=59	40
ＩＮＴ３	71	pm=99	78
ＩＮＴ３	82	pm=119	
ＩＮＴ３	37+ts	ts	167
ＩＮＴ　imm8	30	37	23
ＩＮＴ　imm8	44	pm=59	40
ＩＮＴ　imm8	77	pm=99	78
ＩＮＴ　imm8	86	pm=119	
ＩＮＴ　imm8	37+ts	ts	167
ＩＮＴＯ	Pass:28,Fail:3	Fail:3,pm=3; Pass:35	Fail:3, Pass:24
ＩＮＴＯ	46	pm=59	41
ＩＮＴＯ	73	pm=99	49
ＩＮＴＯ	84	pm=119	
ＩＮＴＯ	39+ts	ts	168
ＩＮＶＤ	4		
ＩＮＶＬＰＧ	12（ヒット時）		
ＩＲＥＴ	15	22,pm=38	17,pm=31
ＩＲＥＴ	36	pm=82	55
ＩＲＥＴ	ts+32	ts	169
ＩＲＥＴＤ	15	22,pm=38	
ＩＲＥＴＤ	36	pm=82	
ＩＲＥＴＤ	15	pm=60	
ＩＲＥＴＤ	ts+32	ts	
ＪＡ　rel8	3/1	7+m,3	7,3

命令	486	386	286
JAE rel8	3/1	7+m,3	7,3
JB rel8	3/1	7+m,3	7,3
JBE rel8	3/1	7+m,3	7,3
JC rel8	3/1	7+m,3	7,3
JCXZ rel8	3/1	9+m,5	8,4
JECXZ rel8	3/1	9+m,5	
JE rel8	3/1	7+m,3	7,3
JZ rel8	3/1	7+m,3	7,3
JG rel8	3/1	7+m,3	7,3
JGE rel8	3/1	7+m,3	7,3
JL rel8	3/1	7+m,3	7,3
JLE rel8	3/1	7+m,3	7,3
JNA rel8	3/1	7+m,3	7,3
JNAE rel8	3/1	7+m,3	7,3
JNA rel8	3/1	7+m,3	7,3
JNAE rel8	3/1	7+m,3	7,3
JNB rel8	3/1	7+m,3	7,3
JNBE rel8	3/1	7+m,3	7,3
JNC rel8	3/1	7+m,3	7,3
JNE rel8	3/1	7+m,3	7,3
JNG rel8	3/1	7+m,3	7,3
JNGE rel8	3/1	7+m,3	7,3
JNL rel8	3/1	7+m,3	7,3
JNLE rel8	3/1	7+m,3	7,3
JNO rel8	3/1	7+m,3	7,3
JNP rel8	3/1	7+m,3	7,3
JNS rel8	3/1	7+m,3	7,3
JNZ rel8	3/1	7+m,3	7,3
JO rel8	3/1	7+m,3	7,3
JP rel8	3/1	7+m,3	7,3
JPE rel8	3/1	7+m,3	7,3
JPO rel8	3/1	7+m,3	7,3
JS rel8	3/1	7+m,3	7,3
JZ rel8	3/1	7+m,3	7,3
JA rel16/32	3/1	7+m,3	
JAE rel16/32	3/1	7+m,3	
JB rel16/32	3/1	7+m,3	
JBE rel16/32	3/1	7+m,3	
JC rel16/32	3/1	7+m,3	
JE rel16/32	3/1	7+m,3	
JZ rel16/32	3/1	7+m,3	

命令	486	386	286
JG rel16/32	3/1	7+m,3	
JGE rel16/32	3/1	7+m,3	
JL rel16/32	3/1	7+m,3	
JLE rel16/32	3/1	7+m,3	
JNA rel16/32	3/1	7+m,3	
JNAE rel16/32	3/1	7+m,3	
JNB rel16/32	3/1	7+m,3	
JNBE rel16/32	3/1	7+m,3	
JNC rel16/32	3/1	7+m,3	
JNE rel16/32	3/1	7+m,3	
JNG rel16/32	3/1	7+m,3	
JNGE rel16/32	3/1	7+m,3	
JNL rel16/32	3/1	7+m,3	
JNLE rel16/32	3/1	7+m,3	
JNO rel16/32	3/1	7+m,3	
JNP rel16/32	3/1	7+m,3	
JNS rel16/32	3/1	7+m,3	
JNZ rel16/32	3/1	7+m,3	
JO rel16/32	3/1	7+m,3	
JP rel16/32	3/1	7+m,3	
JPE rel16/32	3/1	7+m,3	
JPO rel16/32	3/1	7+m,3	
JS rel16/32	3/1	7+m,3	
JZ rel16/32	3/1	7+m,3	
JMP rel8	3/1	7+m,3	7
JMP rel16	3/1	7+m,3	7
JMP r/m16	5/5	7+m/10+m	7/11
JMP ptr16:16	17,pm=19	12+m,pm=27+m	11,pm=23
JMP ptr16:16	32	pm=45+m	38
JMP ptr16:16	42+ts	ts	175
JMP ptr16:16	43+ts	ts	180
JMP m16:16	13,pm=18	43+m,pm=31+m	15,pm=26
JMP m16:16	31	pm=49+m	41
JMP m16:16	41+ts	5+ts	178
JMP m16:16	42+ts	5+ts	183
JMP rel32	3	7+m	
JMP r/m32	5/5	7+m,10+m	
JMP ptr16:32	13,pm=18	12+m,pm=27+m	
JMP ptr16:32	31	pm=45+m	
JMP ptr16:32	42+ts	ts	
JMP ptr16:32	42+ts	ts	

命令	486	386	286
JMP m16:32	13,pm=18	43+m,pm=31+m	
JMP m16:32	31	pm=49+m	
JMP m16:32	41+ts	5+ts	
JMP m16:32	42+ts	5+ts	
LAHF	3	2	2
LAR r16,r/m16	11/11	pm=15/16	14/16
LAR r32,r/m32	11/11	pm=15/16	
LEA r16,m	1	2	3
LEA r32,m	1	2	
LEA r16,m	1	2	
LEA r32,m	1	2	
LEAVE	5	4	5
LEAVE	5	4	
LGDT m16&32	11	11	11
LIDT m16&32	11	11	12
LDS r16,m16;16	6/12	7,pm=22	7,pm=21
LDS r32,m16:32	6/12	7,pm=22	
LSS r16,m16:16	6/12	7,pm=22	
LSS r32,m16:32	6/12	7,pm=22	
LES r16,m16:16	6/12	7,pm=22	7,pm=21
LES r32,m16:16	6/12	7,pm=25	
LFS r32,m16:16	6/12	7,pm=25	
LFS r32,m16:32	6/12	7,pm=25	
LGS r16,m16:16	6/12	7,pm=25	
LGS r32,m16:32	6/12	7,pm=25	
LLDT r/m16	11/11	20	17/19
LMSW r/w16	13/13	10/13	3/6
LOCK	1	0	0
LODS m18	5	5	5
LODS m16	5	5	5
LODS m32	5	5	
LODSB	5	5	5
LODSW	5	5	5
LODSD	5	5	
LOOP rel8	2,6	11+m	8,noj=4
LOOPE rel8	9,6	11+m	8,noj=4
LOOPZ rel8	9,6	11+m	8,noj=4
LOOPNE rel8	9,6	11+m	8,noj=4
LOOPNZ rel8	9,6	11+m	8,noj=4
LSL r16,r/m16	10/10	pm=20/21	14/16

命令	486	386	286
ＬＳＬ　r32,r/m32	10/10	pm=20/2	
ＬＳＬ　r16,r/m16	10/10	pm=25/26	14/16
ＬＳＬ　r32,r/m32	10/10	pm=25/26	
ＬＴＲ　r/m16	20/20	pm=23/27	17/19
ＭＯＶ　r/m8,r8	1	2/2	2/3
ＭＯＶ　r/m16,r16	1	2/2	2/3
ＭＯＶ　r/m32,r32	1	2/2	
ＭＯＶ　r8,r/m8	1	2/4	2/5
ＭＯＶ　r16,r/m16	1	2/4	2/5
ＭＯＶ　r32,r/m32	1	2/4	
ＭＯＶ　r/m16,Sreg	3/3	2/2	2/3
ＭＯＶ　Sreg,r/m16	3/9	2/5,pm=18/19	2/5,pm=17/19
ＭＯＶ　AL,moffs8	1	4	5
ＭＯＶ　AX,moffs16	1	4	5
ＭＯＶ　EAX,moffs32	1	4	
ＭＯＶ　moffs8,AL	1	4	3
ＭＯＶ　moffs,AX	1	2	3
ＭＯＶ　moffs32,EAX	1	2	
ＭＯＶ　reg8,imm8	1	2	2
ＭＯＶ　reg16,imm16	1	2	2
ＭＯＶ　reg32,imm32	1	2	
ＭＯＶ　r/m8,imm8	1	2/2	2/3
ＭＯＶ　r/m16,imm16	1	2/2	2/3
ＭＯＶ　r/m32,imm32	1	2/2	
ＭＯＶ、CRO,r32	16		
ＭＯＶ　r32,CR0/CR2/CR3	4	6	
ＭＯＶ　CR0/CR2/CR3,r32	4	10/4/5	
ＭＯＶ　r32,DR0～3	10	22	
ＭＯＶ　r32,DR6/DR7	10	14	
ＭＯＶ　DR0～3,r32	11	22	
ＭＯＶ　DR6/DR7,r32	11	16	
ＭＯＶ　r32,TR6,TR7	4	12	
ＭＯＶ　TR6/TR7.r32	4	12	
ＭＯＶ、r32,TR3		3	
ＭＯＶＳ　m8,m8	7	7	5
ＭＯＶＳ　m16,m16	7	7	5
ＭＯＶＳ　m32,m32	7	7	
ＭＯＶＳＢ	7	7	5
ＭＯＶＳＷ	7	7	5
ＭＯＶＳＤ	7	7	
ＭＯＶＳＸ　r16,r/m8	3/3	3/6	

命令	486	386	286
MOVSX r32,r/m8	3/3	3/6	
MOVSX r32,r/m16	3/3	3/6	
MOVZX r16,r/m8	3/3	3/6	
MOVZX r32,r/m8	3/3	3/6	
MOVZX r32,r/m16	3/3	3/6	
MUL AL,r/m8	13/18,13/18	9～14/12～17	13/16
MUL AX,r/m16	13/26,13/26	9～22/12～25	21/24
MUL EAX,r/m32	13/42,13/42	9～38/12～41	
NEG r/m8	1/3	2/6	2/7
NEG r/m16	1/3	2/6	2/7
NEG r/m32	1/3	2/6	
NOP	1	3	3
NOT r/m8	1/3	2/6	2/7
NOT r/m16	1/3	2/6	2/7
NOT r／m32	1/3	2/6	2/7
OR AL,imm8	1	2	3
OR AX,imm16	1	2	3
OR EAX,imm32	1	2	
OR r/m8,imm8	1/3	2/7	3/7
OR r/m16,imm16	1/3	2/7	3/7
OR r/m32,imm32	1/3	2/7	
OR r/m16,imm8	1/3	2/7	
OR r/m32,imm8	1/3	2/7	
OR r/m8,r8	1/3	2/6	2/7
OR r/m16,r16	1/3	2/6	2/7
OR r/m32,r8	1/3	2/6	
OR r8,r/m8	1/2	2/7	2/7
OR r16,r/m16	1/2	2/7	2/7
OR r32,r/m32	1/2	2/7	
OUT imm8,AL *	16,pm=11/31, vm=29	10,pm=4/24	3
OUT imm8,AX *	16,pm=11/31, vm=29	10,pm=4/24	3
OUT imm8,EAX *	16,pm=11/31, vm=29	10,pm=4/24	
OUT DX,AL *	16,pm=11/31, vm=29	10,pm=5/25	3
OUT DX,AX *	16,pm=11/31, vm=29	10,pm=5/25	3
OUT DX,EAX *	16,pm=11/31, vm=29	10,pm=5/25	
OUTS DX,r/m8 *	17,pm=10/32, vm=30	14,pm～8/28	5

命令	486	386	286
OUTS DX,r/m16 *	17,pm=10/32, vm=30	14,pm～8/28	5
OUTS DX,r.m32,EAX *	17,pm=10/32, vm=30	14,pm～8/28	
OUTSB *	17,pm=10/32, vm=30	14,pm～8/28	5
OUTSW *	17,pm=10/32, vm=30	14,pm～8/28	5
OUTSD *	17,pm=10/32, vm=30	14,pm～8/28	
POP m16	6	5	5
POP m32	6	5	
POP r16	4	4	5
POP r32	4	4	
POP DS	3	7,pm=21	5,pm=20
POP ES	3	7,pm=21	5,pm=20
POP SS	3	7,pm=21	
POP FS	3	7,pm=21	
POP GS	3	7,pm=21	
POPA	9	24	19
POPAD	9	24	
POPF	9,pm=6	5	5
POPFD	9,pm=6	5	5
PUSH m16	4	5	
PUSH m32	4	5	3
PUSH r16	1	2	
PUSH r32	1	2	3
PUSH imm8	1	2	3
PUSH imm16	1	2	
PUSH imm32	1	2	
PUSH CS	3	2	3
PUSH SS	3	2	3
PUSH DS	3	2	3
PUSH ES	3	2	
PUSH FS	3	2	
PUSH GS	3	2	
PUSHA	11	18	17
PUSHAD	11	18	
PUSHF	4,pm=3	4	3
PUSHFD	4,pm=3	4	
RCL r/m8,1	3/4	9/10	2/7

命令	486	386	286
RCL　r/m8,CL	8～30/9～31	9/10	5/8
RCL　r/m8,imm8	8～30/9～31	9/10	5/8
RCL　r/m16,1	3/4	9/10	2/7
RCL　r/m16,CL	8～30/9～31	9/10	5/8
RCL　r/m16,imm8	8～30/9～31	9/10	5/8
RCL　r/m32,1	3/4	9/10	
RCL　r/m32,CL	8～30/9～31	9/10	
RCL　r/m32,imm8	8～30/9～31	9/10	
RCR　r/m8,1	3/4	9/10	2/7
RCR　r/m8,CL	8～30/9～31	9/10	5/8
RCR　r/m8,imm8	8～30/9～31	9/10	5/8
RCR　r/m16,1	3/4	9/10	2/7
RCR　r/m16,CL	8～30/9～31	9/10	5/8
RCR　r/m16,imm8	8～30/9～31	9/10	5/8
RCR　r/m32,1	3/4	9/10	
RCR　r/m32,CL	8～30/9～31	9/10	
RCR　r/m32,imm8	8～30/9～31	9/10	
ROL　r/m8,1	3/4	3/7	2/7
ROL　r/m8,CL	3/4	3/7	5/8
ROL　r/m8,imm8	2/4	3/7	5/8
ROL　r/m16,1	3/4	3/7	2/7
ROL　r/m16,CL	3/4	3/7	5/8
ROL　r/m16,imm8	2/4	3/7	5/8
ROL　r/m32,1	3/4	3/7	
ROL　r/m32,CL	3/4	3/7	
ROL　r/m32,imm8	2/4	3/7	
ROR　r/m8,1	3/4	3/7	
ROR　r/m8,CL	3/4	3/7	5/8
ROR　r/m8,imm8	2/4	3/7	5/8
ROR　r/m16,1	3/4	3/7	2/7
ROR　r/m16,CL	3/4	3/7	5/8
ROR　r/m16,imm8	2/4	3/7	5/8
ROR　r/m32,1	3/1	3/7	
ROR　r/m32,CL	3/4	3/7	
ROR　r/m32,imm8	2/4	3/7	
REP　INS　r/m8,DX　*	16+8(E)CX, pm=10+8(E)/ 30+8(E)CX, vm=29+8(E)CX	13+6(E)CX, pm=7+6(E)CX/ 27+6(E)CX	5+4CX
REP　INS　r/m16,DX　*	16+8(E)CX, pm=10+8(E)/ 30+8(E)CX, vm=29+8(E)CX	13+6(E)CX, pm=7+6(E)CX/ 27+6(E)CX	5+4CX

命令	486	386	286
REP　INS　r/m32,DX　*	16+8(E)CX, pm=10+8(E)/ 30+8(E)CX, vm=29+8(E)CX	13+6(E)CX, pm=7+6(E)CX/ 27+6(E)CX	
REP　MOVS　m8,m8　*	5,13,12+3(E)CX	5+4(E)CX	5+4CX
REP　MOVS　m16,m16　*	5,13,12+3(E)CX	5+4(E)CX	5+4CX
REP　MOVS　m32,m32　*	5,13,12+3(E)CX	5+4(E)CX	
REP　OUTS　DX,r/m8　*	17+8(E)CX, pm=11+5(E)CX/ 31+5(E)CX	5+12(E)CX, pm=6+5(E)CX/ 26+5(E)CX	5+4CX
REP　OUTS　DX,r/m16　*	17+8(E)CX, pm=11+5(E)CX/ 31+5(E)CX	5+12(E)CX, pm=6+5(E)CX/ 26+5(E)CX	5+4CX
REP　OUTS　DX,r/m32　*	17+8(E)CX, pm=11+5(E)CX/ 31+5(E)CX	5+12(E)CX, pm=6+5(E)CX/ 26+5(E)CX	
REP　LODS　m8　*	5,7+4(E)CX		
REP　LODS　m16　*	5,7+4(E)CX		
REP　LODS　m32　*	5,7+4(E)CX		
REP　STOS　m8　*	5,7+4(E)CX	5+5(E)CX	4+3CX
REP　STOS　m16　*	5,7+4(E)CX	5+5(E)CX	4+3CX
REP　STOS　m32　*	5,7+4(E)CX	5+5(E)CX	
REPE　CMPS　m8,m8　*	5,7+4(E)CX	5+9N	5+9N
REPE　CMPS　m16,m16　*	5,7+4(E)CX	5+9N	5+9N
REPE　CMPS　m32,m32　*	5,7+4(E)CX	5+9N	
REPE　SCAS　m8　*	5,7+4(E)CX	5+8N	5+8N
REPE　SCAS　m16	5,7+4(E)CX	5+8N	5+8N
REPE　SCAS　m32　*	5,7+4(E)CX	5+8N	
REPNE　CMPS　m8,m8　*	5,7+4(E)CX	5+9N	5+9N
REPNE　CMPS　m16,m16*	5,7+4(E)CX	5+9N	5+9N
REPNE　CMPS　m32,32　*	5,7+4(E)CX	5+9N	
REPNE　SCAS　m8　*	5,7+4(E)CX	5+8N	5+8N
REPNE　SCAS　m16　*	5,7+4(E)CX	5+8N	5+8N
REPNE　SCAS　m32　*	5,7+4(E)CX	5+8N	
RET	5	10+m	11
RET	13,pm=18	18+m,pm=32+m	1,pm=25
RET	13,pm=33	pm=68	55
RET　imm16	5	10+m	11
RET　imm16	14,pm=17	18+m,pm=32+m	15,pm=25
RET　imm16	14,pm=33	pm=68	55
SAHF	2	3	2
SAL　r/m8,1	3/4	3/7	2/7
SAL　r/m8,CL	3/4	3/7	5/8

命令	486	386	286
SAL r/m8,imm8	2/4	3/7	5/8
SAL r/m16,1	3/4	3/7	2/7
SAL r/m16,CL	3/4	3/7	5/8
SAL r/m16,imm8	2/4	3/7	5/8
SAL r/m32,1	3/4	3/7	
SAL r/m32,CL	3/4	3/7	
SAL r/m32,imm8	2/4	3/7	
SAR r/m8,1	3/4	3/7	2/7
SAR r/m8,CL	3/4	3/7	5/8
SAR r/m8,imm8	2/4	3/7	5/8
SAR r/m16,1	3/4	3/7	2/7
SAR r/m16,CL	3/4	3/7	5/8
SAR r/m16,imm8	2/4	3/7	5/8
SAR r/m32,1	3/4	3/7	
SAR r/m32,CL	3/4	3/7	
SAR r/m32,imm8	2/4	3/7	
SHL r/m8,1	3/4	3/7	2/7
SHL r/m8,CL	3/4	3/7	5/8
SHL r/m8,imm8	2/4	3/7	5/8
SHL r/m16,1	3/4	3/7	2/7
SHL r/m16,CL	3/4	3/7	5/8
SHL r/m16,imm8	2/4	3/7	5/8
SHL r/m32,1	3/4	3/7	
SHL r/m32,CL	3/4	3/7	
SHL r/m32,imm8	2/4	3/7	
SHR r/m8,1	3/4	3/7	2/7
SHR r/m8,CL	3/4	3/7	5/8
SHR r/m8,imm8	2/4	3/7	5/8
SHR r/m16,1	3/4	3/7	2/7
SHR r/m16,CL	3/4	3/7	5/8
SHR r/m16,imm8	2/4	3/7	5/8
SHR r/m32,1	3/4	3/7	
SHR r/m32,CL	3/4	3/7	
SHR r/m32,imm8	2/4	3/7	
SBB AL,imm8	1	2	3
SBB AX,imm16	1	2	3
SBB EAX,imm32	1	2	
SBB r/m8,imm8	1/3	2/7	3/7
SBB r/m16,imm16	1/3	2/7	3/7
SBB r/m32,imm32	1/3	2/7	
SBB r/m16,imm8	1/3	2/7	3/7

命令	４８６	３８６	２８６
ＳＢＢ　r/m32,imm8	1/3	2/7	
ＳＢＢ　r/m8,r8	1/3	2/6	2/7
ＳＢＢ　r/m16,r16	1/3	2/6	2/7
ＳＢＢ　r/m32,r32	1/3	2/6	
ＳＢＢ　r8,r/m8	1/2	2/7	2/7
ＳＢＢ　r16,r/m16	1/2	2/7	2/7
ＳＢＢ　r32,r/m32	1/2	2/7	
ＳＣＡＳ　m8	6	7	7
ＳＣＡＳ　m16	6	7	7
ＳＣＡＳ　m32	6	7	
ＳＣＡＳＢ	6	7	7
ＳＣＡＳＷ	6	7	7
ＳＣＡＳＤ	6	7	
ＳＥＴＡ　r/n8	4/3	4/5	
ＳＥＴＡＥ　r/n8	4/3	4/5	
ＳＥＴＢ　r/m8	4/3	4/5	
ＳＥＴＢＥ　r/m8	4/3	4/5	
ＳＥＴＣ　r/m8	4/3	4/5	
ＳＥＴＥ　r/m8	4/3	4/5	
ＳＥＴＧ　r/m8	4/3	4/5	
ＳＥＴＧＥ　r/m8	4/3	4/5	
ＳＥＴＧ　r/m8	4/3	4/5	
ＳＥＴＬ　r/m8	4/3	4/5	
ＳＥＴＬＥ　r/m8	4/3	4/5	
ＳＥＴＮＳ　r/m8	4/3	4/5	
ＳＥＴＮＡＥ　r/m8	4/3	4/5	
ＳＥＴＮＢ　r/m8	4/3	4/5	
ＳＥＴＮＢＥ　r/m8	4/3	4/5	
ＳＥＴＮＣ　r/m8	4/3	4/5	
ＳＥＴＮＥ　r/m8	4/3	4/5	
ＳＥＴＮＧ　r/m8	4/3	4/5	
ＳＥＴＮＧＥ　r/m8	4/3	4/5	
ＳＥＴＮＬ　r/m8	4/3	4/5	
ＳＥＴＮＬＥ　r/m8	4/3	4/5	
ＳＥＴＮＯ　r/m8	4/3	4/5	
ＳＥＴＮＰ　r/m8	4/3	4/5	
ＳＥＴＮＳ　r/m8	4/3	4/5	
ＳＥＴＮＺ　r/m8	4/3	4/5	
ＳＥＴＯ　r/m8	4/3	4/5	
ＳＥＴＰ　r/m8	4/3	4/5	
ＳＥＴＰＥ　r/m8	4/3	4/5	

命令	486	386	286
SETPO r/m8	4/3	4/5	
SETS r/m8	4/3	4/5	
SETZ r/m8	4/3	4/5	
SGDT m	10	9	11
SIDT m	10	9	11
SHLD r/m16,r16	2/3	3/7	
SHLD r/m32,r32,imm8	2/3	3/7	
SHLD r/m16,r16,CL	2/3	3/7	
SHLD r/m32,r32,CL	2/3	3/7	
SHRD r/m16,r16,imm8	2/3	3/7	
SHRD r/m32r32,imm8	2/3	3/7	
SHRD r/m16,r16,CL	3/4	3/7	
SHRD r/m32,r32,CL	3/4	3/7	
SLDT r/m16	2/3	2/3,pm=2/2	2/3
SMSW r/m16	2/3	2/3,pm=2/2	2/3
STC	2	2	2
STD	2	2	2
STI	5	3	2
STOS m8	5	4	3
STOS m16	5	4	3
STOS m32	5	4	
STOSB	5	4	3
STOSW	5	4	3
STOSD	5	4	
STR r/m16	2/3	pm=23/27	2/3
SUB AL,imm8	1	2	3
SUB AX,imm16	1	2	3
SUB EAX,imm32	1	2	
SUB r/m8,imm8	1/3	2/7	3/7
SUB r/m16,imm16	1/3	2/7	3/7
SUB r/m32,imm32	1/3	2/7	
SUB r/m16,imm8	1/3	2/7	3/7
SUB r/m32,imm8	1/3	2/7	
SUB r/m8,r8	1/3	2/6	2/7
SUB r/m16,r16	1/3	2/6	2/7
SUB r/m32,r32	1/3	2/6	
SUB r8,r/m8	1/2	2/7	2/7
SUB r16,r/m32	1/2	2/7	2/7
SUB r32,r/m32	1/2	2/7	
TEST AL,imm8	1	2	3
TEST AX,imm8	1	2	3

命令	486	386	286
TEST　EAX,imm32	1	2	
TEST　r/m8,imm8	1/2	2/5	3/6
TEST　r/m16,imm16	1/2	2/5	3/6
TEST　r/m32,immm32	1/2	2/5	
TEST　imm8,r/m8	1/2	2/5	2/6
TEST　imm16,r/m16	1/2	2/5	2/6
TEST　imm32,r/m32	1/2	2/5	
VERR　r/m16	11/11	pm=10/11	14/16
VERW　r/m16	11/11	pm=15/16	14/16
WAIT	1～3	6	3
WBINVD	5		
XADD　r/m8,r8	3/4		
XADD　r/m16,r168	3/4		
XADD　r/m32,r32	3/4		
XCHG　r/m8,r8	3/5	3/5	3/5
XCHG　r8,r/m8	3/5	3/5	3/5
XCHG　r/m16,r16	3/5	3/5	3/5
XCHG　r16,r/m16	3/5	3/5	3/5
XCHG　r/m32,r32	3/5	3/5	
XCHG　r32,r/m32	3/5	3/5	
XCHG　AX,r16	3	3	3
XCHG　r16,AX	3	3	3
XCHG　EAX,r32	3	3	
XCHG　r32,EAX	3	3	
XLAT　m8	4	5	5
XLATB	4	5	5
XOR　AL,imm8	1	2	3
XOR　AX,imm16	1	2	3
XOR　EAX,imm32	1	2	
XOR　r/m8,imm8	1/3	2/7	3/7
XOR　r/m16,imm16	1/3	2/7	3/7
XOR　r/m32,imm32	1/3	2/7	
XOR　r/m16,imm8	1/3	2/7	
XOR　r/m32,imm8	1/3	2/7	
XOR　r/m8,r8	1/3	2/6	2/7
XOR　r/m16,r16	1/3	2/6	2/7
XOR　r/m32,r32	1/3	2/6	
XOR　r8,r/m8	1/2	2/7	2/7
XOR　r16,r/m16	1/2	2/7	2/7
XOR　r32,r/m32	1/2	2/7	

■コプロセッサ命令

●表の読み方

◆命令

ＳＴ	レジスタが現在スタックのトップにある
ＳＴ（ｉ）	トップからｉ（０≦ｉ≧７）番目のスタックエレメント内のレジスタ
ｓｈｏｒｔ	メモリ中のshort実数またはshortバイナリ整数
ｌｏｎｇ	メモリ中のlong実数またはlongバイナリ整数
ｔｅｍｐ	メモリ中のテンポラリ実数
ｐａｃｋ	メモリ中のパック整数
ｗｏｒｄ	メモリ中のワードバイナリ整数
ｎｎバイト	nnバイト長のメモリエリア

命令	486	387	287
F2XM1	242(140～279)	211～476	211～476
FABS	3	22	10～17
FADD　ST,ST(i)/ST(i),ST	10(8～20)	23～34	70～100
FADD　short	10(8～20)	24～32	90～120
FADD　long	10(8～20)	29～37	90～125
FADDP　ST(i),ST	10(8～20)	23～34	75～105
FBLD　pack	75(70～103)	5	290～310
FBSTP　pack	175(172～176)	512～534	520～540+EA
FCHS	6	24～25	10～17
FCLEX／FNCLEX	7	11	2～8
FCOM　//ST(i)	4	24	40～50
FCOM　short	4	26	60～70
FCOM　long	4	31	65～75
FCOMP　//ST(i)	4	26	45～52
FCOMP　short	4	26	63～73
FCOMP　long	4	31	67～77
FCOMPP	5	26	45～55
FCOS	241(193～279)	123～773	
FDECSTP	3	22	6～12
FDIV　//ST(i),ST	73	88～91	193～203
FDIV　short	73	89	215～225
FDIV　long//ST<ST(i)	73	94	200～230
FDIVP　//ST(i),ST	73	88～91	198～209
FDIVR　//ST(i),ST/ST(i)	73	88～91	198～208
FDIVR　short	73	89	215～225
FDIVR　long	73	94	220～230
FDIVRP　ST(i),ST	73	88～91	198～208
FFREE　ST(i)	3	18	9～16
FIADD　word	22.5(19～32)	73～85	102～137
FIADD　short	24(20～35)	57～72	108～143
FICOM　word	18(16～20)	71～75	72～86
FICOM　short	16.5(15～17)	56～63	78～91
FICOMP　word	18(16～20)	71～75	74～88
FICOMP　short	16.5(15～17)	56～63	80～93
FIDIV　word	73	136～140	224～238
FIDIV　short	73	120～127	230～243
FIDIVR　word	73	135～141	224～238
FIDIVR　short	73	121～128	230～243
FILD　word	11.5(9～12)	61～65	46～54
FILD　short	14.5(13～16)	45～52	52～60

命令	486	387	287
FILD long	16.8(10～18)	56～67	60～68
FIMUL word	8	76～87	124～138
FIMUL short	8	61～82	130～144
FINCSTP	3	21	6～12
FINIT／FNINT	17	33	2～8
FIST word	33.4(29～34)	82～95	80～90
FIST short	32.4(28～34)	79～93	82～92
FISTP word	33.4(29～34)	82～95	82～92
FISTP short	33.4(29～34)	79～93	84～94
FISTP long	33.4(29～34)	80～97	94～105
FISUB word	22.5(19～32)	71～83	102～137
FISUB short	24(20～35)	57～82	108～143
FISUBR word	22.5(19～32)	72～84	102～137
FISUBR short	24(20～35)	58～83	108～143
FLD ST(i)	4	14	17～22
FLD short	3	20	38～56
FLD long	3	25	40～60
FLD temp	6	44	53～65
FLDCW 2バイト	4	19	7～14
FLDENV 14バイト	44/34	71	35～45
FLDLG2	8	41	18～24
FLDLN2	8	41	17～23
FLDL2E	8	40	12～21
FLDL2T	8	40	16～22
FLDLPI	8	40	16～22
FLDZ	4	20	11～17
FLD1	4	24	15～21
FMUL //ST(i),ST/ST,ST(1)*	16	29～57	90～145
FMUL //ST(i),ST/ST,ST(1)	16	29～57	90～145
FMUL short	11	27～35	110～125
FMUL long *		32～57	112～168
FMUL long	14	32～57	112～168
FMULP ST(i),ST*		29～57	198～208
FMULP ST(i),ST	16	29～57	198～208
FNOP	3	12	10～16
FPATAN	5(2～17)	314～487	250～800
FPREM	2(2～8)	74～155	15～190
FPREM1	94.5(72～167)	95～185	
FPTAN	244(200～273)	191～573	30～540
FRNDINT	29.1(21～30)	66～80	16～50

命令	486	387	287
FRSTOR 94バイト	131/120	308	205～215
FSAVE／FNSAVE 94バイト		375～376	205～215
FSCALE	31(30～32)	67～86	32～38
FSETPM	2～8	2～8	2～8
FSIN	241(193～279)	122～771	
FSINCOS	291(243～329)	194～809	
FSQRT	85.5(83～87)	122～129	180～186
FST ST(i)	3	11	15～22
FST short	7	44	84～90
FST long	8	45	96～104
FSTCW／FNSTCW 2バイト		15	12～18
FSTENV／FNSTENV 14バイト		103～104	40～50
FSTP ST(i)	3	12	17～24
FSTP short	7	44	86～92
FSTP long	8	45	98～106
FSTP temp	6	53	52～58
FSTSW／FNSTSW 2バイト	3	15	12～18
FSTSW AX／FNSTSW AX	3	13	10～16
FSUB //ST,ST(i)／ST(i),ST	7(5～17)	26～37	70～100
FSUB short	7(5～17)	24～32	90～120
FSUB long	7(5～17)	24～36	95～125
FSUBP ST(i),ST	7(5～17)	26～37	75～105
FSUBR //ST,ST(i)/ST(i),ST	7(5～17)	26～37	70～100
FSUBR short	7(5～17)	25～33	90～120
FSUBR long	7(5～17)	29～37	95～125
FSUBRP ST(i),ST	7(5～17)	26～37	75～105
FTST	4	28	38～48
FUCOM //ST(i)	4	24	
FUCOMP //ST(i)	4	26	
FUCOMPP	5	26	
FWAIT	1～3	3+5n	
FXAM	8	30～38	12～23
FXCH //ST(i)	4	18	10～15
FXTRACT	19(16～20)	70～76	27～55
FYL2X	311(196～329)	120～538	900～1100
FYL2XP1	313(171～326)	257～547	700～1000
F2XM1	242	211～476	310～630

(参考文献　TURBO ASSEMBLER クイックリファレンスガイド)

§ 4-7 MS-DOS ファンクションコール一覧

00H	プログラムの終了
	Call　AH=00H CS=PSP（プログラムセグメントプレフィックス）のセグメントアドレス
	Ret.　なし
01H	エコー付きのキーボード入力 → 入力された文字が画面に出力される.
	Call　AH=01H
	Ret.　AL=入力された文字
02H	文字の出力
	Call　AH=02H DL=スクリーン出力する文字
	Ret.　なし
03H	ＡＵＸ入力（補助入力）
	Call　AH=03H
	Ret.　AL=補助装置から入力された文字
04H	ＡＵＸ出力（補助出力）
	Call　AH=04H DL=補助装置に出力する文字
	Ret.　なし
05H	プリンタ出力
	Call　AH=05H DL=プリンタに出力する文字
	Ret.　なし
06H	直接コンソール入出力
	Call　AH=06H DL=標準出力（スクリーン）に出力する文字（DL≠FFHの場合） FF（入力の場合）
	Ret.　AL=キャラクタ（DL≠FFH，ゼロフラグがセットされている場合） AL=00H（DL≠FFH，ゼロフラグがセットされていない場合） なし（DL=FFHの場合）
07H	直接コンソール入力

	Call　AH=07H
	Ret.　AL=標準入力（キーボード）から入力された文字
08H	キーボード入力
	Call　AH=08H
	Ret.　AL=標準入力（キーボード）から入力された文字
09H	文字列のスクリーン出力
	Call　AH=09H DS:DX='$'で終わり，スクリーンに出力する文字列のポインタ
	Ret.　なし
0AH	バッファ付きのキーボード入力
	Call　AH=0AH DS:DX=入力バッファのポインタ
	Ret.　なし
0BH	キーボードステータスのチェック
	Call　AH=0BH
	Ret.　AL=FFH（タイプアヘッドバッファ内に文字が入っている）， 00H（タイプアヘッドバッファ内に文字が入っていない）
0CH	バッファを空にしてのキーボード入力
	Call　AH=0CH AL=01H,06H,07H,08H,0AH （対応するMS-DOSファンクションリクエストの実行） ほかの値（これ以上の処理は行わない）
	Ret.　AL=00H（タイプアヘッドバッファを空にした）
0DH	ディスクのリセット
	Call　AH=0DH
	Ret.　なし
0EH	ドライブの選択
	Call　AH=0EH DL=ドライブ番号（00H=A:,01H=B:,…）
	Ret.　AL=論理ドライブの数
0FH	ファイルのオープン
	Call　AH=0FH DS:DX=オープンされていないFCB
	Ret.　AL=00H（ディレクトリエントリが存在する） FFH（ディレクトリエントリが存在しない）
10H	ファイルのクローズ
	Call　AH=10H DS:DX=オープンされているFCB

	Ret. AL=00H（ディレクトリエントリが存在する） FFH（ディレクトリエントリが存在しない）
11H	最初のエントリを検索
	Call AH=11H DS:DX=オープンされていないFCB
	Ret. AL=00H（ディレクトリエントリが存在する） FFH（ディレクトリエントリが存在しない）
12H	次のエントリを検索
	Call AH=12H DS:DX=オープンされていないFCB
	Ret. AL=00H（ディレクトリエントリが存在する） FFH（ディレクトリエントリが存在しない）
13H	ファイルの削除
	Call AH=13H DS:DX=オープンされていないFCB
	Ret. AL=00H（ディレクトリエントリが存在する） FFH（ディレクトリエントリが存在しない）
14H	シーケンシャルリード
	Call AH=14H DS:DX=オープンされているFCB
	Ret. AL=00H（正常な読み込み） 01H（EOF） 02H（ディスク転送アドレスで示されるバッファが小さすぎる） 03H（EOF，レコードの一部分）
15H	シーケンシャルライト
	Call AH=15H DS:DX=オープンされているFCB
	Ret. AL=00H（正常な書き込み） 01H（ディスクに空き領域がない） 02H（ディスク転送アドレスで示されるバッファが小さすぎる）
16H	ファイルの作成
	Call AH=16H DS:DX=オープンされていないFCB
	Ret. AL=00H（空のディレクトリが存在する） FFH（空のディレクトリが存在しない）
17H	ファイル名の変更
	Call AH=17H DS:DX=修正されたFCB
	Ret. AL=00H（ディレクトリエントリが存在する） FFH（目的のディレクトリエントリが存在しないか，

	ファイル名がすでに存在する）
19H	カレントドライブの取得
	Call　AH=19H
	Ret.　AL=カレントドライブ番号（00H=A:,01H=B:,…）
1AH	ディスク転送アドレスの設定
	Call　AH=1AH DS:DX=ディスク転送アドレスのポインタ
	Ret.　なし
1BH	カレントドライブデータの取得
	Call　AH=1BH
	Ret.　AL=1クラスタ当たりのセクタ数 CX=1セクタ当たりのバイト数 DX=1ドライブ当たりのクラスタ数 DS:BX=FAT-IDのポインタ
1CH	任意のドライブデータの取得
	Call　AH=1CH Dl=ドライブ番号（00H=A:,01H=B:,…）
	Ret.　AL=FFH（ドライブ番号の指定が無効） =1クラスタ当たりのセクタ数（AL≠FFH） CX=1セクタ当たりのバイト数 DX=1ドライブ当たりのクラスタ数 DS:BX=FAT-IDのポインタ
21H	ランダムリード
	Call　AH=21H DS:DX=オープンされているFCB
	Ret.　AL=00H（正常な読み込み） 01H（EOF） 02H（ディスク転送アドレスで示されるバッファが小さすぎる） 03H（EOF，レコードの一部分）
22H	ランダムライト
	Call　AH=22H DS:DX=オープンされているFCB
	Ret.　AL=00H（正常な書き込み） 01H（ディスクに空き領域がない） 02H（ディスク転送アドレスで示されるバッファが小さすぎる）
23H	ファイルサイズの取得
	Call　AH=23H DS:DX=オープンされていないFCB
	Ret.　AL=00H（ディレクトリエントリが存在する） FFH（ディレクトリエントリが存在しない）

24H	相対レコードの設定
	Call　AH=24H DS:DX=オープンされているFCB
	Ret.　なし
25H	割り込みベクタの設定
	Call　AH=25H AL=割り込みタイプ番号 DS:DX=割り込み処理ルーチンのポインタ
	Ret.　なし
26H	新しいPSPの作成
	Call　AH=26H DX=新しいPSPのポインタ
	Ret.　なし
27H	ランダムブロックリード
	Call　AH=27H DS:DX=オープンされているFCB CX=読み出すレコード数
	Ret.　AL=00H（正常な読み込み） 01H（EOF，空レコード） 02H（ディスク転送アドレスで示されるバッファが小さすぎる） 03H（EOF，レコードの一部分） CX=読み取れたレコード数
28H	ランダムブロックリード
	Call　AH=28H DS:DX=オープンされているFCB CX=00H（ファイルサイズフィールドの設定） 書き込むレコード数（CX≠00H）
	Ret.　AL=00H（正常な書き込み） 01H（ディスクに空き領域がない） 02H（ディスク転送アドレスで示されるバッファが小さすぎる） CX=書き込めたレコード数
29H	ファイルネームの解析
	Call　AH=29H AL=解析の制御 DS:SI=解析するストリング ES:DI=オープンされていないFCB
	Ret.　AL=00H（ワイルドカードが使用されていない） 01H（ワイルドカードが使用されている） FFH（ドライブ文字が無効） DS:SI=解析したストリングの直後のアドレス ES:DI=オープンされていないFCB
2AH	日付の取得

	Call　AH=2AH
	Ret.　CX=年（1980～2099） DH=月（1～12） DL=日（1～31） AL=曜日（00H=日，01H=月，…06H=土）
2BH	日付の設定
	Call　AH=2BH CX=年（1980～2099） DH=月（1～12） DL=日（1～31）
	Ret.　AL=00H（有効な日付） FFH（無効な日付）
2CH	時刻の取得
	Call　AH=2CH
	Ret.　CH=時（0～23） CL=分（0～59） DH=秒（0～59）
2DH	時刻の設定
	Call　AH=2DH CH=時（0～23） CL=分（0～59） DH=秒（0～59） DL=00H
	Ret.　AL=00H（有効な時刻） FFH（無効な時刻）
2EH	ベリファイフラグの制御
	Call　AH=2EH AL=00H（ベリファイを行わない） 01H（ベリファイを行う） DL=00H
	Ret.　なし
2FH	ディスクの転送アドレスの取得
	Call　AH=2FH
	Ret.　ES:BX=ディスク転送アドレスのポインタ
30H	ＤＯＳのバージョンの取得
	Call　AH=30H
	Ret.　AL=バージョン番号の整数部 AH=バージョン番号の小数部 BH=OEMのシリアル番号 BL:CX=24ビットのユーザーシリアル番号（OEMによって異なる）
31H	プログラムの常駐終了（キーププロセス）

	Call	AH=31H AL=抜け出しコード DX=パラグラフ（16バイト単位）でのメモリサイズ
	Ret.	なし
3300H	ブレーク（CTRL-C）チェックの制御	
	Call	AH=33H AL=00H（状態の取得） 01H（設定） DL=00H（AL=01Hの場合，CTRL-C検査の解除） 01H（AL=01Hの場合，CTRL-C検査の設定）
	Ret.	DL=00H（AL=00Hの場合，CTRL-C検査が設定されていない） 01H（AL=00Hの場合，CTRL-C検査が設定されている） AL=FFHでエラー
3305H	ブートドライブ番号の取得	
	Call	AH=3305H
	Ret.	DL=ドライブ番号（01H=A:,02H=B:,…）
3306H	DOSの配置情報の取得	
	Call	AH=3306H
	Ret.	BL=バージョン番号の整数部 BH=バージョン番号の小数部 DH=MS-DOSのバージョンフラグ 10H（MS-DOSはハイメモリにある） DL=下位の３ビットにリビジョン番号，ほかのビットは０
35H	割り込みベクタの取得	
	Call	AH=35H AL=割り込み番号
	Ret.	ES:BX=割り込みルーチンのポインタ
36H	ディスクの空き容量の取得	
	Call	AH=36H DL=ドライブ番号（00H=カレント,01H=A:,02H=B:,…）
	Ret.	BX=使用可能なクラスタ数 DX=１ドライブ当たりのクラスタ数 CX=１セクタ当たりのバイト数 AX=１クラスタ当たりのセクタ数（AX≠FFFFH） FFFFH（ドライブ番号が無効）
38H	国別情報の取得または設定	
	Call	AH=38H AL=情報の取得国のカントリーコード （00H=現在の国，01H=USA，51H=日本） FFH（２バイトのカントリーコードの設定のとき） BX=カントレコード（AL=FFHの場合） DS:DX=情報に対する32バイトのバッファのポインタ

	Ret.	CF=0（DS:DXで指定したバッファに情報が設定された） 1　（AX=02H，無効なカントリーコードが指定された）
	Call	AH=38H DX=FFFFH AL=FFH（2バイトのカントリーコードを使用するとき） 　カントリーコード（AL≠FFH） BX=カントリーコード（AL=FFHの場合）
	Ret.	CF=0（エラーなし） 1　（AX=02H，無効なカントリーコードが指定された）
39H	ディレクトリの作成	
	Call	AH=39H DS:DX=パス名を示すASCIZ文字列のポインタ
	Ret.	CF=0（エラーなし） CF=1 AX=03H（無効なパス） 　05H（アクセス拒否）
3AH	ディレクトリの削除	
	Call	AH=3AH DS:DX=パス名を示すASCIZ文字列のポインタ
	Ret.	CF=0（エラーなし） 1　AX=03H（無効なパス） 　05H（アクセスの否定） 　10H（カレントディレクトリ）
3BH	カレントディレクトリの変更	
	Call	AH=3BH DS:DX=パス名を示すASCIZ文字列のポインタ
	Ret.	CF=0（エラーなし） 1　AX=03H（無効なパス）
3CH	ファイルの作成	
	Call	AH=3CH DS:DX=パス名を示すASCIZ文字列のポインタ CX=ファイルの属性
	Ret.	CF=0　AX=ファイルハンドル 1　AX=03H（無効なパス） 　04H（オープンされているファイルが多すぎる） 　05H（アクセスの否定）
3DH	ファイルのオープン	
	Call	AH=3DH AL=ファイルアクセスコントロール DS:DX=パス名を示すASCIZ文字列のポインタ
	Ret.	CF=0　AX=ファイルハンドル 1　AX=01H（無効なファンクションコード） 　02H（ファイルが存在しない）

	03H（無効なパス） 04H（オープンしているファイルが多すぎる） 05H（アクセスの否定） 0CH（無効なアクセス）
3EH	ファイルのクローズ
	Call　AH=3EH BX=ファイルハンドル
	Ret.　CF=0（エラーなし） 1　AX=06H（無効なハンドル）
3FH	ファイルまたはデバイスのリード
	Call　AH=3FH DS:DX=バッファのポインタ CX=読み込むバイト数 BX=ファイルハンドル
	Ret.　CF=0　AX=読み込まれたバイト数 1　AX=05H（アクセスできない） 06H（ハンドルが無効）
40H	ファイルまたはデバイスのライト
	Call　AH=40H DS:DX=バッファのポインタ CX=書き込むバイト数 BX=ファイルハンドル
	Ret.　CF=0　AX=書き込まれたバイト数 1　AX=05H（アクセスできない） 06H（ハンドルが無効）
41H	ディレクトリエントリの削除
	Call　AH=41H DS:DX=パス名を示すASCIZ文字列のポインタ
	Ret.　CF=0（エラーなし） 1　AX=02H（無効なファイル） 05H（アクセスの否定）
42H	ファイルポインタの移動
	Call　AH=42H CX:DX=オフセット（移動するバイト数） AL=00H（ファイルの先頭からオフセットを加えた位置に移動） 01H（現在の位置からオフセットを加えた位置に移動） 02H（ファイルの終端にオフセットを加えた位置に移動） BX=ファイルハンドル
	Ret.　CF=0　DX:AX=新規のポインタロケーション 1　AX=01H（無効なファンクション） 06H（無効なハンドル）
43H	アトリビュートの取得または設定
	Call　AH=43H

	DS:DX=パス名を示すASCIZ文字列のポインタ AL=00H（属性の取得） 　01H（属性を設定する） CX=設定する属性（AL=01Hの場合）
	Ret.　CF=0　CX=属性（AL=00Hの場合） 　1　AX=01H（無効なファンクション） 　　02H（無効なファイル名） 　　03H（無効なパス） 　　05H（アクセスの否定）
4400H	IOCTLデータの取得
	Call　AX=4400H BX=ハンドル
	Ret.　CF=0　DX=デバイスデータ 　1　AX=01H（無効なファンクション） 　　06H（無効なハンドル）
4401H	IOCTLデータの設定
	Call　AX=4401H BX=ハンドル DX=デバイスデータ
	Ret.　CF=0　DX=デバイスデータ 　1　AX=01H（無効なファンクション） 　　06H（無効なハンドル）
4402H	IOCTLキャラクタの受け取り
	Call　AX=4402H BX=ハンドル CX=コントロールデータのバイト数 DS:DX=バッファのポインタ
	Ret.　CF=0　AX=転送されたバイト数 　1　AX=01H（無効なファンクション） 　　06H（無効なハンドル）
4403H	IOCTLキャラクタの送信
	Call　AX=4403H BX=ハンドル CX=コントロールデータのバイト数 DS:DX=バッファのポインタ
	Ret.　CF=0　AX=転送されたバイト数 　1　AX=01H（無効なファンクション） 　　06H（無効なハンドル）
4404H	IOCTLブロックの受け取り
	Call　AX=4404H BL=ドライブ番号（00H=カレント，01H=A:，02H=B:…） CX=コントロールデータのバイト数 DS:DX=バッファのポインタ

	Ret. CF=0 AX=転送されたバイト数 1 AX=01H（無効なファンクション） 05H（無効なドライブ番号）
4405H	IOCTLブロックの送信
	Call AX=4405H BL=ドライブ番号（00H=カレント，01H=A:，02H=B:…） CX=コントロールデータのバイト数 DS:DX=バッファのポインタ
	Ret. CF=0 AX=転送されたバイト数 1 AX=01H（無効なファンクション） 05H（無効なドライブ番号）
4406H	入力ステータスのチェック
	Call AX=4406H BX=ハンドル
	Ret. CF=0 AL=00H（レディ状態ではない） FFH（レディ） 1 AX=01H（無効なファンクション） 05H（アクセスが否定されました） 06H（無効なハンドル） 0DH（無効なデータ）
4407H	出力ステータスのチェック
	Call AX=4407H BX=ハンドル
	Ret. CF=0 AL=00H（レディ状態ではない） FFH（レディ） 1 AX=01H（無効なファンクション） 05H（アクセスが否定されました） 06H（無効なハンドル） 0DH（無効なデータ）
4408H	IOCTLの交換性
	Call AX=4408H BL=ドライブ番号（00H=カレント，01H=A:，02H=B:…）
	Ret. CF=0 AX=00H（交換可能） 01H（交換不可能） 1 AX=01H（無効なファンクション） 0FH（無効なドライブ番号）
4409H	IOCTLリディレクトブロック
	Call AX=4409H BL=ドライブ番号（00H=カレント，01H=A:，02H=B:…）
	Ret. CF=0 DX=デバイスアトリビュートワード 1 AX=01H（無効なファンクション） 0FH（無効なドライブ番号）
440AH	IOCTLリディレクトハンドル

	Call	AX=440AH BX=ハンドル
	Ret.	CF=0 DX=IOCTLビットフィールド 1 AX=01H（無効なファンクションコード） 06H（無効なハンドル）
440BH	IOCTLリトライ	
	Call	AX=440BH DX=リトライの回数 CX=待ち時間
	Ret.	CF=0（エラーなし） 1 AX=01H（無効なファンクションコード）
440CH	一般IOCTL（ハンドル用）	
	Call	AX=440CH BX=ハンドル CH=05H（カテゴリーコード） CL=ファンクション（マイナー）コード DS:DX=データバッファへのポインタ
	Ret.	CF=0（エラーなし） 1 AX=01H（無効なファンクションコード）
440DH	一般IOCTL（ブロックデバイス用）	
	Call	AX=440DH BL=デバイス番号（00H=カレント，01H=A:，02H=B:…） CH=08H（カテゴレコード） CL=ファンクション（マイナー）コード DS:DX=パラメータブロック-1へのポインタ
	Ret.	CF=0（エラーなし） 1 AX=01H（無効なファンクションコード） 02H（無効なドライブ）
440EH	論理ドライブマップの取得	
	Call	AX=440EH BX=ドライブ番号（00H=カレント，01H=A:，02H=B:…）
	Ret.	CF=0 AL=00H（物理的にマップされている） 1 AX=01H（無効なファンクションコード） 0FH（無効なドライブ番号）
440FH	論理ドライブマップの設定	
	Call	AX=440FH BX=ドライブ番号（00H=カレント，01H=A:，02H=B:…）
	Ret.	CF=0 AL=00H（物理的にマップされた） 1 AX=01H（無効なファンクションコード） 0FH（無効なドライブ番号）
4410H	プリンタサポートの問い合わせ	
	Call	AX=4410H

		BX=ファイルハンドル CH=05H CL=ファンクションコード (45H：繰返カウントの設定/65H：繰返カウントの取得)
	Ret.	CF=0（エラーなし） 1　AX=01H（無効なファンクションコード）
4411H	IOCTLデバイスの問い合わせ	
	Call	AX=4411H BX=ドライブ番号（00H=カレント，01H=A:，02H=B:…） CH=08H CL=ファンクションコード（★多数）
	Ret.	CF=0（エラーなし） 1　AX=01H（無効なファンクションコード）
45H	ファイルハンドルの二重化	
	Call	AH=45H BX=ファイルハンドル
	Ret.	CF=0　AX=新規のファイルハンドル 1　AX=04H（オープンされているファイルが多すぎる） 06H（無効なハンドル）
46H	ファイルハンドルの強制二重化	
	Call	AH=46H BX=既存のファイルハンドル CX=新規のファイルハンドル
	Ret.	CF=0（エラーなし） 1　AX=04H（オープンされているファイルが多すぎる） 06H（無効なハンドル）
47H	カレントディレクトリの取得	
	Call	AH=47H DS:SI=64バイトのメモリ領域に対するポインタ DL=ドライブ番号（00H=カレント，01H=A:，02H=B:…）
	Ret.	CF=0（エラーなし） 1　AX=0FH（無効なドライブ）
48H	メモリの割り当て	
	Call	AH=48H BX=割り当てるメモリサイズ（パラグラフ）
	Ret.	CF=0 AX=割り当てられたメモリのセグメントアドレス（パラグラフ） 1　AX=07H（メモリ中のデータの破壊） 08H（十分なサイズのメモリがない） BX=割り当て可能なメモリサイズ
49H	割り当てメモリの解放	
	Call	AH=49H

	ES=解放するメモリのセグメントアドレス
	Ret. CF=0（エラーなし） 1 AX=07H（メモリ中のデータの破壊） 09H（無効なブロック）
4AH	割り当てメモリブロックの変更
	Call AH=4AH ES=変更するメモリのセグメントアドレス BX=変更するメモリのサイズ（パラグラフ）
	Ret. CF=0（エラーなし） 1 AX=07H（メモリ中のデータの破壊） 08H（十分なサイズのメモリがない） 09H（無効なブロック）
4B00H	プログラムのロードと実行
	Call AX=4B00H DS:DX=パス名（ASCIZ文字列）のポインタ ES:BX=パラメータブロックのポインタ
	Ret. CF=0（エラーなし） 1 AX=01H（無効なファンクション） 02H（ファイルが存在しない） 04H（オープンされているファイルが多すぎる） 05H（アクセスの否定） 08H（十分な大きさのメモリがない） 0AH（不正な環境） 0BH（不正なフォーマット）
4B03H	オーバーレイロード
	Call AX=4B03H DS:DX=パス名（ASCIZ文字列）のポインタ ES:BX=パラメータブロックのポインタ
	Ret. CF=0（エラーなし） 1 AX=01H（無効なファンクション） 02H（ファイルが存在しない） 04H（オープンされているファイルが多すぎる） 05H（アクセスの否定） 0AH（不正な環境）
4B05H	新しいプログラムの実行準備
	Call AX=4B05H DS:DX=EXECSTATE構造体のポインタ
	Ret. なし
4CH	プロセスの終了
	Call AH=4CH AL=リターンコード
	Ret. なし

4DH	子プロセスのリターンコード取得
	Call　AH=4DH
	Ret.　AX=抜け出しコード
4EH	最初に一致するファイル名の検索
	Call　AH=4EH DS:DX=パス名（ASCIZ文字列）のポインタ CX=属性
	Ret.　CF=0（エラーなし） 1　AX=02H（パス名が無効） 12H（ファイルがない）
4FH	つぎに一致するファイル名の検索
	Call　AH=4FH
	Ret.　CF=0（エラーなし） 1　AX=12H（ファイルがない）
54H	ベリファイの状態の取得
	Call　AH=54H
	Ret.　AL=00H（ベリファイオフ） 01H（ベリファイオン）
56H	ディレクトリエントリの変更
	Call　AH=56H DS:DX=既存ファイルのパス名（ASCIZ文字列）のポインタ ES:DI=新規のパス名のポインタ
	Ret.　CF=0（エラーなし） 1　AX=02H（ファイルが存在しない） 05H（アクセスの拒否） 11H（装置の不一致）
57H	ファイルのタイムスタンプの取得または設定
	Call　AH=57H AL=00H（取得する） 01H（設定する） BX=ファイルハンドル CX=設定する時刻（AL=01Hの場合） DX=設定する日付（AL=01Hの場合）
	Ret.　CF=0（エラーなし） CX=時刻（取得時） DX=日付（取得時） 1　AX=01H（無効なファンクション） 06H（無効なハンドル）
5800H	アローケーションストラテジの取得
	Call　AX=5800H
	Ret.　CF=0　AX=アロケーションストラテジの値

	1　AX=01H（無効なファンクションコード）
5801H	アローケーションストラテジの設定
	Call　AX=5801H BX=アロケーションストラテジ
	Ret.　CF=0（エラーなし） 1　AX=01H（無効なファンクションコード）
5802H	上位メモリブロックのリンク状態の取得
	Call　AX=5802H
	Ret.　AL=00H（上位メモリブロックがリンクされていない） 01H（上位メモリブロックがリンクされている）
5803H	上位メモリブロックのリンク状態の設定
	Call　AX=5803H BX=00H（上位メモリブロックがリンクしない） 01H（上位メモリブロックがリンクする）
	Ret.　CF=0（エラーなし） 1　AX=01H（無効なファンクション） 07H（アリーナが不正）
59H	拡張されたエラーコードの取得
	Call　AH=59H BX=00H
	Ret.　AX=拡張されたエラーコード BH=エラークラス BL=可能な対処 CH=ローカス CL,DX,SI,DI,BP,DS,ESの各レジスタの内容は破壊される
5AH	テンポラリファイルの作成
	Call　AH=5AH CX=アトリビュート DS:DX=パス名（ASCIZ文字列）のポインタ
	Ret.　CF=0　AX=ファイルハンドル 1　AX=03H（パス名が存在しない） 05H（アクセスができない）
5BH	新しいファイルの作成
	Call　AH=5BH CX=アトリビュート DS:DX=パス名（ASCIZ文字列）のポインタ
	Ret.　CF=0　AX=ファイルハンドル 1　AX=03H（パスが存在しない） 04H（オープンするファイルが多すぎる） 05H（アクセスできない） 50H（ファイルがすでに存在する）

番号		内容
5C00H	ファイルアクセスのロック	
	Call	AX=5C00H BX=ファイルハンドル CX:DX=ファイルのロックされた領域のポインタ SI:DI=領域のサイズ
	Ret.	CF=0（エラーなし） 1　AX=01H（無効なファンクションコード） 06H（無効なハンドル） 21H（ロックの破壊）
5C01H	ファイルアクセスのロック解除	
	Call	AX=5C01H BX=ハンドル CX:DX=ロックを解除する領域のポインタ SI:DI=領域のサイズ
	Ret.	CF=0（エラーなし） 1　AX=01H（無効なファンクションコード） 06H（無効なハンドル） 21H（ロックの破壊）
5E00H	マシン名の取得	
	Call	AX=5E00H DS:DX=バッファのポインタ
	Ret.	CF=0　CX=ローカルコンピュータの番号 1　AX=01H（無効なファンクションコード）
5E02H	プリンタのセットアップ	
	Call	AX=5E02H BX=割り当てリストのインデックス CX=セットアップ文字列の長さ DS:SI=セットアップ文字列のポインタ
	Ret.	CF=0（エラーなし） 1　AX=01H（無効なファンクションコード）
5F02H	割り当てリストエントリの取得	
	Call	AX=5F02H BX=割り当てリストのインデックス DS:SI=ローカル名のためのバッファのポインタ ES:DI=リモート名のためのバッファのポインタ
	Ret.	CF=0　BL=03H（プリンタ） 04H（ドライブ） CX=ユーザー変数域 1　AX=01H（無効なファンクションコード） 12H（これ以上のファイルはない）
5F03H	割り当てリストエントリの作成	
	Call	AX=5F03H BL=03H（プリンタ）

	04H（ドライブ） CX=ユーザー変数域 DS:SI=ソースデバイス名のポインタ ES:DI=ディスティネーションデバイス名のポインタ
	Ret. CF=0（エラーなし） 1 AX=01H（無効なファンクションコード） 03H（パスが見つからない） 05H（アクセスできない） 08H（メモリ不足）
5F04H	割り当てリストエントリの解除
	Call AX=5F04H DS:SI=ソースデバイス名のポインタ
	Ret. CF=0（エラーなし） 1 AX=01H（無効なファンクションコード） 0FH（サーバ上のリディレクトの中止）
62H	PSPの取得
	Call AH=62H
	Ret. BX=カレントプロセスのセグメントアドレス
67H	オープン可能な最大ハンドル数の設定
	Call AH=67H BX=設定する最大ハンドル数
	Ret. なし
68H	ファイルのコミット
	Call AH=68H BX=フラッシュするファイルのハンドル
	Ret. CF=0（エラーなし） 1 AX=02H（ファイルが見つからない）
6CH	拡張されたファイルのオープン／クリエイト
	Call AH=6CH BX=ファイルをオープンするときのモード CX=ファイルのアトリビュート DX=アクション DS:SI=パス名の位置
	Ret. CF=0（エラーなし） 1 AX=03H（無効なパス） 04H（オープンされているファイルが多すぎる） 05H（アクセスの否定）

§ 4-8 エスケープシーケンス一覧表

	エスケープシーケンス	機　能
カーソル移動	ESC [*pl*;*pc* H	カーソルを *pl* 行 *pc* 桁の位置に移動させる *pl* の値が最終行の値より大きい時は，最終行に設定される *pl* の値が0，あるいは省略された時は，1行目に設定される *pc* の値が最終桁の値より大きい時は，最終桁に設定される *pc* の値が0，あるいは省略された時は，1桁目に設定される
	ESC [*pl*;*pc* f	ESC [*pl*;*pc* H と同じ機能を持つ
	ESC= *lc*	ESC [*pl* ;pc H と同じ機能を持つ *l* とcはそれぞれ *pl* , *pc* に対応するが,省略は不可能 尚，*l* と *c* には20Hのオフセット値が加えられた値を設定する
	ESC [*pn* A	カーソルを同じ桁の位置で上に *pn* 行移動させる カーソルが先頭行にある時，あるいは先頭行を越えた時は先頭行に設定される *pn* の値が0，あるいは省略された時は，*pn* の値は1に設定される
	ESC [*pn* B	カーソルを同じ桁の位置で下に *pn* 行移動させる カーソルが最終行にある時，あるいは最終行を越えた時は最終行に設定される *pn* の値が0，あるいは省略された時は，*pn* の値は1に設定される
	ESC [*pn* C	カーソルを右に *pn* 文字移動させる カーソルが右端にある時，あるいは右端を越えたとき，右端に設定される *pn* の値が0，あるいは省略された時は，*pn* の値は1に設定される
	ESC [*pn* D	カーソルを左に *pn* 文字移動させる カーソルが左端にある時，あるいは左端を越えたとき，左端に設定される *pn* の値が0，あるいは省略された時は，*pn* の値は1に設定される
	ESC D	カーソルを同じ桁の位置で，1行下に移動させる カーソルが最終行にある時は，1行スクロールアップさせる
	ESC E	カーソルを1行下の左端に移動させる カーソルが最終行にある時は，1行スクロールアップさせる
	ESC M	カーソルを同じ桁の位置で1行上に移動させる カーソルが先頭行にある時は，1行スクロールダウンさせる
	ESC [s	カーソル位置(行，桁)とその表示文字の属性をセーブする
	ESC [u	ESC [s でセーブした内容を戻す 以前に ESC [s が実行されていない時は，カーソルをホームポジションに移動させ，属性は既定値となる
	ESC [6n	カーソルの位置を，そのすぐ後のコンソール入力呼び出しで知らせる その形式は　ESC [*pl* ; *pc* R

文字・行の削除／入力	ESC [0J	カーソルの位置から，最終行の右端までをクリアする カーソルの位置は移動しない パラメータの0は省略可能
	ESC [1J	ホームポジションからカーソル位置までをクリアする カーソルの位置は移動しない
	ESC [2J	画面をクリアしてカーソルをホームポジションにセットする
	ESC *	ESC [2J と同じ機能を持つ
	ESC [0K	カーソルの位置からカーソル行の右端までをクリアする カーソルの位置は移動しない パラメータの0は省略可能
	ESC [1K	カーソル行の左端からカーソルの位置までクリアする カーソルの位置は移動しない
	ESC [2k	カーソル行をクリアする カーソルの位置は移動しない
	ESC [*pn* L	カーソル行以降をpn行下に移動し，*pn* 行文のスペースを挿入する カーソルは挿入行の先頭左端に設定される 挿入行が最終行を越えた時、あるいは移動する行が最終行を越えた時は，その越えた分は失われる *pn* の値が0,あるいは省略された時は， *pn* の値は1に設定される
	ESC [*pn* M	カーソル行から下 *pn* 行を削除し，それ以降の行を上につめる カーソルは詰められた行の左端に設定される 最終行を越えての削除は不可能 *pn* の値が0，あるいは省略された時は， *pn* の値は1に設定される
画面モードの設定	ESC)0	漢字を取り扱うモードにする このモードになっていないと漢字を表示することが出来ない 尚，このモードではグラフ文字を取り扱うことは出来ない，システムの既定値はこのモード
	ESC)3	グラフ文字を取り扱うモードにする 尚，このモードでは漢字を取り扱うことは出来ない
	ESC [>5l	画面上にカーソル表示を行うモードにする システムの既定値はこのモード
	ESC [>5h	画面上にカーソル表示を行わないモードにする
	ESC [>1h	ファンクションキーの表示を取り消して，画面の最下行を使用可能にする
	ESC [>1l	画面の最下行に，ファンクションキーの表示をする
	ESC [>3h ESC [>3n	画面の表示行数を20行にするノーマルモードのみ使用可） 画面の表示行数を，31行にする（ハイレゾモードのみ使用可）
	ESC [>3l	画面の表示行数を，25行にするシステムの既定値はこのモード
キーボード割当	ESC [*pn* ;・・・; *pn* p	ＥＳＣに続く最初の１文字に対応するキーに，２番目以降の文字，または文字列を割り当てる
	ESC ["*String*";P	同上
	ESC [*pn* ;"*String*" ;*pn* n	同上

文字・行の削除／入力	ESC [*ps*;･･･;*ps* m	表示文字の属性を指定する 一度指定すれば，属性を変更するまで以降に続く表示文字に有効	
		psの値	内容
		0	規定の属性
		1 2 4 5 7 8，１６ ３０ ３１，１７ ３２，２０ ３３，２１ ３４，１８ ３５，１９ ３６，２２ ３７ ４０ ４１ ４２ ４３ ４４ ４５ ４６ ４７	ハイライト（モノクロのみ） バーチカルライン アンダーライン ブリンク リバース シークレット 黒　　　淡（暗） 赤 緑 黄色 青 紫 水色 白　　　濃（明） リバース黒 リバース赤 リバース緑 リバース黄色 リバース青 リバース紫 リバース水色 リバース白

コード		名称	内容
10進	16進		
0	００	ＮＵＬ	
1	０１	ＴＣ1（ＳＯＨ）	ヘディング開始
2	０２	ＴＣ2（ＳＴＸ）	テキスト開始
3	０３	ＴＣ3（ＥＴＸ）	テキスト終了
4	０４	ＴＣ4（ＥＯＴ）	伝送終了
5	０５	ＴＣ5（ＥＮＱ）	問い合わせ
6	０６	ＴＣ6（ＡＣＫ）	肯定応答
7	０７	ＢＥＬ	ブザーを約１秒間だけ鳴らす
8	０８	ＦＥ0（ＢＳ）	後退（Ｂａｃｋ　Ｓｐａｃｅ） カーソルを１文字分だけ左に移動させる　尚，カーソルが行の左端にある時は１行上の右端に移動し，ホーム位置（行と桁の先頭）にある時は何もしない
9	０９	ＦＥ1（ＨＴ）	水平タブ カーソルを次のタブ位置に移動させる タブの値は半角８文字に決められている 尚，カーソルが72桁目より右にある時は１行下の左端へ移動し，最終行の時は１行分だけスクロールアップする

10	0A	FE2（LF）	改行 カーソルを同じ桁の位置で1行分だけ下に移動させる なお，カーソルが最終行にある時は1行分だけスクロールアップする
11	0B	FE3（VT）	垂直タブ カーソルを同じ桁位置で1行分だけ上に移動させる 尚，カーソルガ先頭行にある時は何もしない
12	0C	FE4（FF）	書式送り カーソルを1文字分だけ右に移動する なお，カーソルが行の右端にある時は1行下の左端に移動し，最終行の左端にある時は1行分だけスクロールアップして左端に移動する
13	0D	FE5（CR）	復帰（Carriage Return） カーソルを行の左端へ移動させる
14	0E	SO	シフト・アウト
15	0F	SI	シフト・イン
16	10	TC7（DLE）	伝送制御拡張
17	11	DC1	装置制御1
18	12	DC2	制御装置2
19	13	DC3	制御装置3
20	14	DC4	制御装置4
21	15	TC8（NAK）	否定応答
22	16	TC9（SYN）	同期信号
23	17	TC10（ETB）	伝送ブロック終了
24	18	CAN	取り消し
25	19	EM	媒体終端（End of Medium）
26	1A	SUB	置換キャラクタ CRT画面をクリアして，カーソルをホームポジションへ移動させる
27	1B	ESC	拡張
28	1C	IS4（FS）	ファイル分離キャラクタ
29	1D	IS5（GS）	グループ分離キャラクタ
30	1E	IS6（RS）	レコード分離キャラクタ
31	1F	IS7（US）	ユニット分離キャラクタ

§ 4-9 プリンタ制御コード表

主なコードのみ示します．ビットイメージや，ダウンロード文字関係のコードは省略してあります．

■プリンタの機種に依存しない標準的な制御コード

機能	コード
印字	０ＤＨ
改行	０ＡＨ
改頁	０ＣＨ
バッファクリア	１８Ｈ

■ESC/P系（*n* には，数値を指定します．）

機能	コード	備考
プリンタの初期化	[ESC]@	
インチ単位ページ長設定	[ESC]C0 *n*	*n* =ページ長
行単位ページ長設定	[ESC]C *n*	*n* = 1 ページの行数
１／８インチ改行モード	[ESC]0	
１／６インチ改行モード	[ESC]2	
n／１８０インチ改行モード	[ESC]3 *n*	*n* =改行量
１／１８０インチ改行	[ESC]J *n*	*n* =改行量
プロポーショナルモード指定，解除	[ESC]p*n*	*n* =1:指定　*n* =0:解除
１０ＣＰＩ指定	[ESC]P	
１２ＣＰＩ指定	[ESC]M	
１５ＣＰＩ指定	[ESC]g	
スーパー／サブスクリプト指定	[ESC]S *n*	*n* =1:サブ　*n* =0:スーパー
スーパー／サブスクリプト解除	[ESC]T	
縮小指定	[SI]	
縮小解除	[DC2]	
自動解除付き横倍指定	[SO]	
自動解除付き横倍解除	[DC4]	
横倍指定	[ESC]W *n*	*n* =1:指定　*n* =0:解除
縦倍指定／解除	[ESC]w *n*	*n* =1:指定　*n* =0:解除
強調指定	[ESC]E	
強調解除	[ESC]F	
２重印字指定	[ESC]G	
２重印字解除	[ESC]H	
アンダーライン指定／解除	[ESC]−*n*	*n* =1:指定　*n* =0:解除
文字間スペース指定	[ESC][SP] *n*	*n* =スペース量

漢字モード指定	[FS]&	
漢字モード解除	[FS].	
漢字縦書き指定	[FS]J	
漢字横書き指定	[FS]K	
半角文字指定	[FS]SI	
半角文字解除	[FS][DC2]	
1／4角文字指定	[FS]rn	*n*=1:指定　*n*=0:解除
4倍角指定／解除	[FS]W *n*	*n*=1:指定　*n*=0:解除
漢字横倍指定／解除	[ESC]W *n*	*n*=1:指定　*n*=0:解除
漢字縦倍指定／解除	[FS]W *n*	*n*=1:指定　*n*=0:解除
漢字アンダーライン指定／解除	[FS]- *n*	*n*=1:指定　*n*=0:解除
全角文字スペース量設定	[FS]Tnn	nn=左右のスペースをそれぞれ nに入れる

■PR201系（*n* には，数値を指定します）

機能	コード	
プリンタの初期化	[ESC] c 1	
コンデンスモード指定	[ESC]Q	
エリートモード指定	[ESC]E	
プロポーショナルモード指定	[ESC]P	
横書き漢字モード指定	[ESC]K	
縦書き漢字モード指定	[ESC]t	
スーパースクリプト文字指定	[ESC]s1	
サブスクリプト文字指定	[ESC]s2	
スクリプト解除	[ESC]s0	
倍角文字の指定	[ESC]e *nn*	*nn*=11:標準　*nn*=12:横倍角 *nn*=21:縦倍角　*nn*=22: 4倍角
強調文字指定	[ESC]!	
強調文字解除	[ESC]"	
アンダーライン指定	[ESC]X	
アンダーライン解除	[ESC]Y	
ドットスペース指定	[ESC] *n*	*n*=スペース量
漢字文字幅3／20インチ指定	[FS]A	
漢字文字幅1／5インチ指定	[FS]B	
漢字文字幅1／6インチ指定	[FS]C	
漢字文字幅2／15インチ指定	[FS]D	
1／6インチ改行モード指定	[ESC]A	
1／8インチ改行モード指定	[ESC]B	
n／120インチ改行モード指定	[ESC]T	

●漢字コード表の見方

行の数字に列の数字を加えたものがその漢字のコードとなります．

例）コード表で一番初めの漢字”亜”ならＪＩＳコードは３０２０＋１＝３０２１，
　シフトＪＩＳなら８８９Ｅ＋１＝８８９Ｆとなります．

●機種依存の文字について

記号などは機種により存在しないものがあります．一般的にはあまり問題となりませんが，コンピューター通信や他機種でファイルを扱う場合などにおいては以下の文字が問題となる場合があります．

・記号（2）　［ＪＩＳコード２２３０～２２７Ｆ］
・罫線　　　［ＪＩＳコード２８２０～２８４Ｆ］
　ＮＥＣ製のＰＣ－９８０１シリーズでは存在しません．
　（ＥＰＳＯＮ製の９８互換機には存在します．）

・記号（3）　［ＪＩＳコード２Ｄ２０～２Ｄ７Ｆ］
　ＰＣ９８シリーズ（ＥＰＳＯＮ互換機も含む）のみ存在します．
　よって通信上や他機種でこれらの文字を含むファイルを扱う際には注意してください．

・拡張漢字　［ＪＩＳコード７９２０～７Ｃ７Ｆ］
　古い機種ではサポートされていません．

	区点	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	シフトJIS	区点
	JIS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
記号(1)	2120		Sp	、	。	，	．	・	：	；	？	！	゛	゜	´	｀	¨	813F	0100
	2130	＾	￣	＿	ヽ	ヾ	ゝ	ゞ	〃	仝	々	〆	〇	ー	―	‐	／	814F	0116
	2140	＼	～	∥	｜	…	‥	‘	’	“	”	（	）	〔	〕	［	］	815F	0132
	2450	｛	｝	〈	〉	《	》	「	」	『	』	【	】	＋	－	±	×	816F	0148
	2160	÷	＝	≠	＜	＞	≦	≧	∞	∴	♂	♀	°	′	″	℃	￥	8180	0164
	2170	＄	¢	£	％	＃	＆	＊	＠	§	☆	★	○	●	◎	◇		8190	0180
	2220		◆	□	■	△	▲	▽	▼	※	〒	→	←	↑	↓	〓		819E	0200
記号(2)	2230											∈	∋	⊆	⊇	⊂	⊃	81AE	0216
	2240	∪	∩									∧	∨	¬	⇒	⇔	∀	81BE	0232
	2250	∃												∠	⊥	⌒	∂	81CE	0248
	2260	∇	≡	≒	≪	≫	√	∽	∝	∵	∫	∬						81DE	0264
	2270			Å	‰	♯	♭	♪	†	‡	¶				◯			81EE	0280
英数字	2330	0	1	2	3	4	5	6	7	8	9							824F	0316
	2340		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	825F	0332
	2350	P	Q	R	S	T	U	V	W	X	Y	Z						826F	0348
	2360		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	8280	0364
	2370	p	q	r	s	t	u	v	w	x	y	z						8290	0380
ひらがな	2420		ぁ	あ	ぃ	い	ぅ	う	ぇ	え	ぉ	お	か	が	き	ぎ	く	829E	0400
	2430	ぐ	け	げ	こ	ご	さ	ざ	し	じ	す	ず	せ	ぜ	そ	ぞ	た	82AE	0416
	2440	だ	ち	ぢ	っ	つ	づ	て	で	と	ど	な	に	ぬ	ね	の	は	82BE	0432
	2450	ば	ぱ	ひ	び	ぴ	ふ	ぶ	ぷ	へ	べ	ぺ	ほ	ぼ	ぽ	ま	み	82CE	0448
	2460	む	め	も	ゃ	や	ゅ	ゆ	ょ	よ	ら	り	る	れ	ろ	ゎ	わ	82DE	0464
	2470	ゐ	ゑ	を	ん													82EE	0480
カタカナ	2520		ァ	ア	ィ	イ	ゥ	ウ	ェ	エ	ォ	オ	カ	ガ	キ	ギ	ク	833F	0500
	2530	グ	ケ	ゲ	コ	ゴ	サ	ザ	シ	ジ	ス	ズ	セ	ゼ	ソ	ゾ	タ	834F	0516
	2540	ダ	チ	ヂ	ッ	ツ	ヅ	テ	デ	ト	ド	ナ	ニ	ヌ	ネ	ノ	ハ	835F	0532
	2550	バ	パ	ヒ	ビ	ピ	フ	ブ	プ	ヘ	ベ	ペ	ホ	ボ	ポ	マ	ミ	836F	0548
	2560	ム	メ	モ	ャ	ヤ	ュ	ユ	ョ	ヨ	ラ	リ	ル	レ	ロ	ヮ	ワ	8380	0564
	2570	ヰ	ヱ	ヲ	ン	ヴ	ヵ	ヶ										8390	0580
ギリシャ文字	2620		Α	Β	Γ	Δ	Ε	Ζ	Η	Θ	Ι	Κ	Λ	Μ	Ν	Ξ	Ο	839F	0600
	2630	Π	Ρ	Σ	Τ	Υ	Φ	Χ	Ψ	Ω								83AE	0616
	2640		α	β	γ	δ	ε	ζ	η	θ	ι	κ	λ	μ	ν	ξ	ο	83BE	0632
	2650	π	ρ	σ	τ	υ	φ	χ	ψ	ω								83CE	0648
ロシア語	2720		А	Б	В	Г	Д	Е	Ё	Ж	З	И	Й	К	Л	М	Н	843F	0700
	2730	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	844F	0716
	2740	Ю	Я															845F	0732
	2750		а	б	в	г	д	е	ё	ж	з	и	й	к	л	м	н	846F	0748

注：Sp は空白（スペース）コード

注：（網掛け部分）はEPSON製互換機のみ存在

	区 点	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		区 点
	JIS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	シフトJIS	
ロシア語	2760	о	п	р	с	т	у	ф	х	ц	ч	ш		ъ	ы	ь	э	8480	0764
	2770	ю	я															8490	0780
罫線	2820		─	│	┌	┐	┘	└	├	┬	┤	┴	┼	━	┃	┏	┓	849E	0800
	2830	┛	┗	┣	┳	┫	┻	╋	┠	┯	┨	┷	┿	┝	┰	┥	┸	84AE	0816
	2840	╂																84BE	0832
半角英数字	2920		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	853F	0900
	2930	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	854F	0916
	2940	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	855F	0932
	2950	P	Q	R	S	T	I	V	W	X	U	X	[	¥	]	^	_	856F	0948
	2960	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	8580	0964
	2970	p	q	r	s	t	u	v	w	x	y	z	{	¦	}	‾	◦	8590	0980
半角カタカナ	2A20		｡	｢	｣	､	･	ｦ	ｧ	ｨ	ｩ	ｪ	ｫ	ｬ	ｭ	ｮ	ｯ	859E	1000
	2A30	ｰ	ｱ	ｲ	ｳ	ｴ	ｵ	ｶ	ｷ	ｸ	ｹ	ｺ	ｻ	ｼ	ｽ	ｾ	ｿ	85AE	1016
	2A40	ﾀ	ﾁ	ﾂ	ﾃ	ﾄ	ﾅ	ﾆ	ﾇ	ﾈ	ﾉ	ﾊ	ﾋ	ﾌ	ﾍ	ﾎ	ﾏ	85BE	1032
	2A50	ﾐ	ﾑ	ﾒ	ﾓ	ﾔ	ﾕ	ﾖ	ﾗ	ﾘ	ﾙ	ﾚ	ﾛ	ﾜ	ﾝ	ﾞ	ﾟ	85CE	1048
	2A60	ヰ	ヱ	ヮ	ヵ	ヶ	ｳﾞ	ｶﾞ	ｷﾞ	ｸﾞ	ｹﾞ	ｺﾞ	ｻﾞ	ｼﾞ	ｽﾞ	ｾﾞ	ｿﾞ	85DE	1064
	2A70	ﾀﾞ	ﾁﾞ	ﾂﾞ	ﾃﾞ	ﾄﾞ	ﾊﾞ	ﾋﾞ	ﾌﾞ	ﾍﾞ	ﾎﾞ	ﾊﾟ	ﾋﾟ	ﾌﾟ	ﾍﾟ	ﾎﾟ		85EE	1080
半角罫線	2B20			″	〃	─	━	│	┃	┄	┅	┆	┇	┈	┉	┊	┋	863F	1100
	2B30	┌	┍	┎	┏	┐	┑	┒	┓	└	┕	┖	┗	┘	┙	┚	┛	864F	1116
	2B40	├	┝	┞	┟	┠	┡	┢	┣	┤	┥	┦	┧	┨	┩	┪	┫	865F	1132
	2B50	┬	┭	┮	┯	┰	┱	┲	┳	┴	┵	┶	┷	┸	┹	┺	┻	866F	1148
	2B60	┼	┽	┾	┿	╀	╁	╂	╃	╄	╅	╆	╇	╈	╉	╊	╋	8680	1164
記号半角	2B70	′	¨	‘	“	〔	〕	〈	〉	《	》	『	』	【	】	—		8690	1180
全角罫線	2C20					─	━	│	┃	┄	┅	┆	┇	┈	┉	┊	┋	869E	1200
	2C30	┌	┍	┎	┏	┐	┑	┒	┓	└	┕	┖	┗	┘	┙	┚	┛	86AE	1216
	2C40	├	┝	┞	┟	┠	┡	┢	┣	┤	┥	┦	┧	┨	┩	┪	┫	86BE	1232
	2C50	┬	┭	┮	┯	┰	┱	┲	┳	┴	┵	┶	┷	┸	┹	┺	┻	86CE	1248
	2C60	┼	┽	┾	┿	╀	╁	╂	╃	╄	╅	╆	╇	╈	╉	╊	╋	86DE	1264
記号(3)	2D20		①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪	⑫	⑬	⑭	⑮	876F	1300
	2D30	⑯	⑰	⑱	⑲	⑳	Ⅰ	Ⅱ	Ⅲ	Ⅳ	Ⅴ	Ⅵ	Ⅶ	Ⅷ	Ⅸ	Ⅹ		874F	1316
	2D40	㍉	㌔	㌢	㍍	㌘	㌧	㌃	㌶	㍑	㍗	㌍	㌦	㌣	㌫	㍊	㌻	875F	1332
	2D50	㎜	㎝	㎞	㎎	㎏	㏄	㎡										876F	1348
	2D60	〝	〟	№	㏍	℡	㊤	㊥	㊦	㊧	㊨	㈱	㈲	㈹	㍾	㍽	㍼	8780	1364
	2D70	≒	≡	∫	∮	Σ	√	⊥	∠	∟	⊿	∵	∩	∪				8790	1380
あ	3020		亜	唖	娃	阿	哀	愛	挨	姶	逢	葵	茜	穐	悪	握	渥	879E	1600

	区点	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		区点
	JIS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	シフトJIS	
あ	3030	旭	葦	芦	鯵	梓	圧	斡	扱	宛	姐	虻	飴	絢	綾	鮎	或	88AE	1616
	3040	粟	袷	安	庵	按	暗	案	闇	鞍	杏							88BE	1632
い												以	伊	位	依	偉	囲		
	3050	夷	委	威	尉	惟	意	慰	易	椅	為	畏	異	移	維	緯	胃	88CE	1648
	3060	萎	衣	謂	違	遺	医	井	亥	域	育	郁	磯	一	壱	溢	逸	88DE	1664
	3070	稲	茨	芋	鰯	允	印	咽	員	因	姻	引	飲	淫	胤	蔭		88EE	1680
	3120		院	陰	隠	韻	吋											893F	1700
う								右	宇	烏	羽	迂	雨	卯	鵜	窺	丑		
	3130	碓	臼	渦	嘘	唄	欝	蔚	鰻	姥	厩	浦	瓜	閏	噂	云	運	394F	1716
	3140	雲																395F	1732
え			荏	餌	叡	営	嬰	影	映	曳	栄	永	泳	洩	瑛	盈	穎		
	3150	頴	英	衛	詠	鋭	液	疫	益	駅	悦	謁	越	閲	榎	厭	円	896F	1748
	3160	園	堰	奄	宴	延	怨	掩	援	沿	演	炎	焔	煙	燕	猿	縁	8980	1764
	3170	艶	苑	薗	遠	鉛	鴛	塩										8990	1780
お									於	汚	甥	凹	央	奥	往	応			
	3220		押	旺	横	欧	殴	王	翁	襖	鴬	鴎	黄	岡	沖	荻	億	899E	1800
	3230	屋	憶	臆	桶	牡	乙	俺	卸	恩	温	穏	音					89AE	1816
か														下	化	仮	何		
	3240	伽	価	佳	加	可	嘉	夏	嫁	家	寡	科	暇	果	架	歌	河	89BE	1832
	3250	火	珂	禍	禾	稼	箇	花	苛	茄	荷	華	菓	蝦	課	嘩	貨	89CE	1848
	3260	迦	過	霞	蚊	俄	峨	我	牙	画	臥	芽	蛾	賀	雅	餓	駕	89DE	1864
	3270	介	会	解	回	塊	壊	廻	快	怪	悔	恢	懐	戒	拐	改		89EE	1880
	3320		魁	晦	械	海	灰	界	皆	絵	芥	蟹	開	階	貝	凱	劾	8A3F	1900
	3330	外	咳	害	崖	慨	概	涯	碍	蓋	街	該	鎧	骸	浬	馨	蛙	3A4F	1916
	3340	垣	柿	蛎	鈎	劃	嚇	各	廓	拡	撹	格	核	殻	獲	確	穫	8A5F	1932
	3350	覚	角	赫	較	郭	閣	隔	革	学	岳	楽	額	顎	掛	笠	樫	8A6F	1948
	3360	橿	梶	鰍	潟	割	喝	恰	括	活	渇	滑	葛	褐	轄	且	鰹	8A80	1964
	3370	叶	椛	樺	鞄	株	兜	竃	蒲	釜	鎌	噛	鴨	栢	茅	萱		8A90	1980
	3420		粥	刈	苅	瓦	乾	侃	冠	寒	刊	勘	勧	巻	喚	堪	姦	8A9E	2000
	3430	完	官	寛	干	幹	患	感	慣	憾	換	敢	柑	桓	棺	款	歓	8AAE	2016
	3440	汗	漢	澗	潅	環	甘	監	看	竿	管	簡	緩	缶	翰	肝	艦	8ABE	2032
	3450	莞	観	諌	貫	還	鑑	間	閑	関	陥	韓	館	舘	丸	含	岸	8ACE	2048
	3460	巌	玩	癌	眼	岩	翫	贋	雁	頑	顔	願						8ADE	2064
き													企	伎	危	喜	器		
	3470	基	奇	嬉	寄	岐	希	幾	忌	揮	机	旗	既	期	棋	棄		8AEE	2080
	3520		機	帰	毅	気	汽	畿	祈	季	稀	紀	徽	規	記	貴	起	8B3F	2100
	3530	軌	輝	飢	騎	鬼	亀	偽	儀	妓	宜	戯	技	擬	欺	犠	疑	8B4F	2116
	3540	祇	義	蟻	誼	議	掬	菊	鞠	吉	吃	喫	桔	橘	詰	砧	杵	8B5F	2132
	3550	黍	却	客	脚	虐	逆	丘	久	仇	休	及	吸	宮	弓	急	救	8B6F	2148
	3560	朽	求	汲	泣	灸	球	究	窮	笈	級	糾	給	旧	牛	去	居	8B80	2164

	区 点	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	シフトJIS	区 点
	JIS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
き	3570	巨	拒	拠	挙	渠	虚	許	距	鋸	漁	禦	魚	亨	享	京		8B90	2118
	3620		供	侠	僑	兇	競	共	凶	協	匡	卿	叫	喬	境	峡	強	8B9E	2200
	3630	彊	怯	恐	恭	挟	教	橋	況	狂	狭	矯	胸	脅	興	蕎	郷	8BAE	2216
	3640	鏡	響	饗	驚	仰	凝	尭	暁	業	局	曲	極	玉	桐	粁	僅	8BBE	2232
	3650	勤	均	巾	錦	斤	欣	欽	琴	禁	禽	筋	緊	芹	菌	衿	襟	8BCE	2248
	3660	謹	近	金	吟	銀												8BDE	2264
く							九	倶	句	区	狗	玖	矩	苦	躯	駆	駈		
	3670	駒	具	愚	虞	喰	空	偶	寓	遇	隅	串	櫛	釧	屑	屈		8BEE	2280
	3720		掘	窟	沓	靴	轡	窪	熊	隈	粂	栗	繰	桑	鍬	勲	君	8C3F	2300
	3730	薫	訓	群	軍	郡												8C4F	2316
け							卦	袈	祁	係	傾	刑	兄	啓	圭	珪	型		
	3740	契	形	径	恵	慶	慧	憩	掲	携	敬	景	桂	渓	畦	稽	系	8C5F	2332
	3750	経	継	繋	罫	茎	荊	蛍	計	詣	警	軽	頚	鶏	芸	迎	鯨	8C6F	2348
	3760	劇	戟	撃	激	隙	桁	傑	欠	決	潔	穴	結	血	訣	月	件	8C80	2364
	3770	倹	倦	健	兼	券	剣	喧	圏	堅	嫌	建	憲	懸	拳	捲		8C90	2380
	3820		検	権	牽	犬	献	研	硯	絹	県	肩	見	謙	賢	軒	遣	8C9E	2400
	3830	鍵	険	顕	験	鹸	元	原	厳	幻	弦	減	源	玄	現	絃	舷	8CAE	2416
	3840	言	諺	限														8CBE	2432
こ					乎	個	古	呼	固	姑	孤	己	庫	弧	戸	故	枯		
	3850	湖	狐	糊	袴	股	胡	菰	虎	誇	跨	鈷	雇	顧	鼓	五	互	8CCE	2448
	3860	伍	午	呉	吾	娯	後	御	悟	梧	檎	瑚	碁	語	誤	護	醐	8CDE	2464
	3870	乞	鯉	交	佼	侯	候	倖	光	公	功	効	勾	厚	口	向		8CEE	2480
	3920		后	喉	坑	垢	好	孔	孝	宏	工	巧	巷	幸	広	庚	康	8D3F	2500
	3930	弘	恒	慌	抗	拘	控	攻	昂	晃	更	杭	校	梗	構	江	洪	8D4F	2516
	3940	浩	港	溝	甲	皇	硬	稿	糠	紅	紘	絞	綱	耕	考	肯	肱	8D5F	2532
	3950	腔	膏	航	荒	行	衡	講	貢	購	郊	酵	鉱	砿	鋼	閤	降	8D6F	2548
	3960	項	香	高	鴻	剛	劫	号	合	壕	拷	濠	豪	轟	麹	克	刻	8D80	2564
	3970	告	国	穀	酷	鵠	黒	獄	漉	腰	甑	忽	惚	骨	狛	込		8D90	2580
	3A20		此	頃	今	困	坤	墾	婚	恨	懇	昏	昆	根	梱	混	痕	8D9E	2600
	3A30	紺	艮	魂														8DAE	2616
さ					些	佐	叉	唆	嵯	左	差	査	沙	瑳	砂	詐	鎖		
	3A40	裟	坐	座	挫	債	催	再	最	哉	塞	妻	宰	彩	才	採	栽	8DBE	2632
	3A50	歳	済	災	采	犀	砕	砦	祭	斎	細	菜	裁	載	際	剤	在	8DCE	2648
	3A60	材	罪	財	冴	坂	阪	堺	榊	肴	咲	崎	埼	碕	鷺	作	削	8DDE	2664
	3A70	咋	搾	昨	朔	柵	窄	策	索	錯	桜	鮭	笹	匙	冊	刷		8DEE	2680
	3B20		察	拶	撮	擦	札	殺	薩	雑	皐	鯖	捌	錆	鮫	皿	晒	8E3F	2700
	3B30	三	傘	参	山	惨	撒	散	桟	燦	珊	産	算	纂	蚕	讃	賛	8E4F	2716
	3B40	酸	餐	斬	暫	残												8E5F	2732
し							仕	仔	伺	使	刺	司	史	嗣	四	士	始		
	3B50	姉	姿	子	屍	市	師	志	思	指	支	孜	斯	施	旨	枝	止	8E6F	2748

	区 点	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		区 点
	JIS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	シフトJIS	
	3B60	死	氏	獅	祉	私	糸	紙	紫	肢	脂	至	視	詞	詩	試	誌	8E80	2764
	3B70	諮	資	賜	雌	飼	歯	事	似	侍	児	字	寺	慈	持	時		8E90	2780
	3C20		次	滋	治	爾	璽	痔	磁	示	而	耳	自	蒔	辞	汐	鹿	8E9E	2800
	3C30	式	識	鴫	竺	軸	宍	雫	七	叱	執	失	嫉	室	悉	湿	漆	9EAE	2816
	3C40	疾	質	実	蔀	篠	偲	柴	芝	屡	蕊	縞	舎	写	射	捨	赦	8EBE	2832
	3C50	斜	煮	社	紗	者	謝	車	遮	蛇	邪	借	勺	尺	杓	灼	爵	8ECE	2848
	3C60	酌	釈	錫	若	寂	弱	惹	主	取	守	手	朱	殊	狩	珠	種	8EDE	2864
	3C70	腫	趣	酒	首	儒	受	呪	寿	授	樹	綬	需	囚	収	周		8EEE	2880
	3D20		宗	就	州	修	愁	拾	洲	秀	秋	終	繍	習	臭	舟	蒐	8F3F	2900
	3D30	衆	襲	讐	蹴	輯	週	酋	酬	集	醜	什	住	充	十	従	戎	8F4F	2916
	3D40	柔	汁	渋	獣	縦	重	銃	叔	夙	宿	淑	祝	縮	粛	塾	熟	8F5F	2932
	3D50	出	術	述	俊	峻	春	瞬	竣	舜	駿	准	循	旬	楯	殉	淳	8F6F	2948
し	3D60	準	潤	盾	純	巡	遵	醇	順	処	初	所	暑	曙	渚	庶	緒	8F80	2964
	3D70	署	書	薯	藷	諸	助	叙	女	序	徐	恕	鋤	除	傷	償		8F90	2980
	3E20		勝	匠	升	召	哨	商	唱	嘗	奨	妾	娼	宵	将	小	少	8F9E	3000
	3E30	尚	庄	床	廠	彰	承	抄	招	掌	捷	昇	昌	昭	晶	松	梢	8FAE	3016
	3E40	樟	樵	沼	消	渉	湘	焼	焦	照	症	省	硝	礁	祥	称	章	8FBE	3032
	3E50	笑	粧	紹	肖	菖	蒋	蕉	衝	裳	訟	証	詔	詳	象	賞	醤	8FCE	3048
	3E60	鉦	鍾	鐘	障	鞘	上	丈	丞	乗	冗	剰	城	場	壌	嬢	常	8FDE	3064
	3E70	情	擾	条	杖	浄	状	畳	穣	蒸	譲	醸	錠	嘱	埴	飾		8FEE	3080
	3F20		拭	植	殖	燭	織	職	色	触	食	蝕	辱	尻	伸	信	侵	903F	3100
	3F30	唇	娠	寝	審	心	慎	振	新	晋	森	榛	浸	深	申	疹	真	904F	3116
	3F40	神	秦	紳	臣	芯	薪	親	診	身	辛	進	針	震	人	仁	刃	905F	3132
	3F50	塵	壬	尋	甚	尽	腎	訊	迅	陣	靭							906F	3148
												笥	諏	須	酢	図	厨		
す	3F60	逗	吹	垂	帥	推	水	炊	睡	粋	翠	衰	遂	酔	錐	錘	随	9080	3164
	3F70	瑞	髄	崇	嵩	数	枢	趨	雛	据	杉	椙	菅	頗	雀	裾		9090	3180
	4020		澄	摺	寸													909E	3200
						世	瀬	畝	是	凄	制	勢	姓	征	性	成	政		
	4030	整	星	晴	棲	栖	正	清	牲	生	盛	精	聖	声	製	西	誠	90AE	3216
	4040	誓	請	逝	醒	青	静	斉	税	脆	隻	席	惜	戚	斥	昔	析	90BE	3232
せ	4050	石	積	籍	績	脊	責	赤	跡	蹟	碩	切	拙	接	摂	折	設	90CE	3248
	4060	窃	節	説	雪	絶	舌	蝉	仙	先	千	占	宣	専	尖	川	戦	90DE	3264
	4070	扇	撰	栓	栴	泉	浅	洗	染	潜	煎	煽	旋	穿	箭	線		90EE	3280
	4120		繊	羨	腺	舛	船	薦	詮	賎	践	選	遷	銭	銑	閃	鮮	913F	3300
	4130	前	善	漸	然	全	禅	繕	膳	糎								914F	3316
											噌	塑	岨	措	曾	曽	楚		
	4140	狙	疏	疎	礎	祖	租	粗	素	組	蘇	訴	阻	遡	鼠	僧	創	915F	3332
そ	4150	双	叢	倉	喪	壮	奏	爽	宋	層	匝	惣	想	捜	掃	挿	掻	916F	3348
	4160	操	早	曹	巣	槍	槽	漕	燥	争	痩	相	窓	糟	総	綜	聡	9180	3364

	区点	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		区点
	JIS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	シフトJIS	
そ	4170	草	荘	葬	蒼	藻	装	走	送	遭	鎗	霜	騒	像	増	憎		9190	3380
そ	4220		臓	蔵	贈	造	促	側	則	即	息	捉	束	測	足	速	俗	919E	3400
そ	4230	属	賊	族	続	卒	袖	其	揃	存	孫	尊	損	村	遜			91AE	3416
た																他	多		
た	4240	太	汰	詑	唾	堕	妥	惰	打	柁	舵	楕	陀	駄	騨	体	堆	91BE	3432
た	4250	対	耐	岱	帯	待	怠	態	戴	替	泰	滞	胎	腿	苔	袋	貸	91CE	3448
た	4260	退	逮	隊	黛	鯛	代	台	大	第	醍	題	鷹	滝	瀧	卓	啄	91DE	3464
た	4270	宅	托	択	拓	沢	濯	琢	託	鐸	濁	諾	茸	凧	蛸	只		91EE	3480
た	4320		叩	但	達	辰	奪	脱	巽	竪	辿	棚	谷	狸	鱈	樽	誰	923F	3500
た	4330	丹	単	嘆	坦	担	探	旦	歎	淡	湛	炭	短	端	箪	綻	耽	924F	3516
た	4340	胆	蛋	誕	鍛	団	壇	弾	断	暖	檀	段	男	談				925F	3532
ち															値	知	地		
ち	4350	弛	恥	智	池	痴	稚	置	致	蜘	遅	馳	築	畜	竹	筑	蓄	926F	3548
ち	4360	逐	秩	窒	茶	嫡	着	中	仲	宙	忠	抽	昼	柱	注	虫	衷	9280	3564
ち	4370	註	酎	鋳	駐	樗	瀦	猪	苧	著	貯	丁	兆	凋	喋	寵		9290	3580
ち	4420		帖	帳	庁	弔	張	彫	徴	懲	挑	暢	朝	潮	牒	町	眺	929E	3600
ち	4430	聴	脹	腸	蝶	調	諜	超	跳	銚	長	頂	鳥	勅	捗	直	朕	92AE	3616
ち	4440	沈	珍	賃	鎮	陳												92BE	3632
つ							津	墜	椎	槌	追	鎚	痛	通	塚	栂	掴		
つ	4450	槻	佃	漬	柘	辻	蔦	綴	鍔	椿	潰	坪	壷	嬬	紬	爪	吊	92CE	3648
つ	4460	釣	鶴															92DE	3664
て				亭	低	停	偵	剃	貞	呈	堤	定	帝	底	庭	廷	弟		
て	4470	悌	抵	挺	提	梯	汀	碇	禎	程	締	艇	訂	諦	蹄	逓		92EE	3680
て	4520		邸	鄭	釘	鼎	泥	摘	擢	敵	滴	的	笛	適	鏑	溺	哲	933F	3700
て	4530	徹	撤	轍	迭	鉄	典	填	天	展	店	添	纏	甜	貼	転	顛	934F	3716
て	4540	点	伝	殿	澱	田	電											935F	3732
と								兎	吐	堵	塗	妬	屠	徒	斗	杜	渡		
と	4550	登	菟	賭	途	都	鍍	砥	砺	努	度	土	奴	怒	倒	党	冬	936F	3748
と	4560	凍	刀	唐	塔	塘	套	宕	島	嶋	悼	投	搭	東	桃	梼	棟	9380	3764
と	4570	盗	淘	湯	涛	灯	燈	当	痘	祷	等	答	筒	糖	統	到		9390	3780
と	4620		董	蕩	藤	討	謄	豆	踏	逃	透	鐙	陶	頭	騰	闘	働	939E	3800
と	4630	動	同	堂	導	憧	撞	洞	瞳	童	胴	萄	道	銅	峠	鴇	匿	93AE	3816
と	4640	得	徳	涜	特	督	禿	篤	毒	独	読	栃	橡	凸	突	椴	届	93BE	3832
と	4650	鳶	苫	寅	酉	瀞	噸	屯	惇	敦	沌	豚	遁	頓	呑	曇	鈍	93CE	3848
な	4660	奈	那	内	乍	凪	薙	謎	灘	捺	鍋	楢	馴	縄	畷	南	楠	93DE	3864
な	4670	軟	難	汝														93EE	3880
に					二	尼	弐	迩	匂	賑	肉	虹	廿	日	乳	入			
に	4720		如	尿	韮	任	妊	忍	認									943F	3900
ぬ										濡									
ね											禰	祢	寧	葱	猫	熱	年		

	区点	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		区点
	JIS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	シフトJIS	
ね	4730	念	捻	撚	燃	粘												944F	3916
の							乃	廼	之	埜	嚢	悩	濃	納	能	脳	膿		
	4740	農	覗	蚤														945F	3932
は					巴	把	播	覇	杷	波	派	琶	破	婆	罵	芭	馬		
	4750	俳	廃	拝	排	敗	杯	盃	牌	背	肺	輩	配	倍	培	媒	梅	946F	3948
	4760	楳	煤	狽	買	売	賠	陪	這	蝿	秤	矧	萩	伯	剥	博	拍	9480	3964
	4770	柏	泊	白	箔	粕	舶	薄	迫	曝	漠	爆	縛	莫	駁	麦		9490	3980
	4820		函	箱	硲	箸	肇	筈	櫨	幡	肌	畑	畠	八	鉢	溌	発	949E	4000
	4830	醗	髪	伐	罰	抜	筏	閥	鳩	噺	塙	蛤	隼	伴	判	半	反	94AE	4016
	4840	叛	帆	搬	斑	板	氾	汎	版	犯	班	畔	繁	般	藩	販	範	94BE	4032
	4850	釆	煩	頒	飯	挽	晩	番	盤	磐	蕃	蛮						94CE	4048
ひ													匪	卑	否	妃	庇		
	4860	彼	悲	扉	批	披	斐	比	泌	疲	皮	碑	秘	緋	罷	肥	被	94DE	4064
	4870	誹	費	避	非	飛	樋	簸	備	尾	微	枇	毘	琵	眉	美		94EE	4080
	4920		鼻	柊	稗	匹	疋	髭	彦	膝	菱	肘	弼	必	畢	筆	逼	953F	4100
	4930	桧	姫	媛	紐	百	謬	俵	彪	標	氷	漂	瓢	票	表	評	豹	954F	4116
	4940	廟	描	病	秒	苗	錨	鋲	蒜	蛭	鰭	品	彬	斌	浜	瀕	貧	955F	4132
	4950	賓	頻	敏	瓶													956F	4148
ふ						不	付	埠	夫	婦	富	冨	布	府	怖	扶	敷		
	4960	斧	普	浮	父	符	腐	膚	芙	譜	負	賦	赴	阜	附	侮	撫	9580	4164
	4970	武	舞	葡	蕪	部	封	楓	風	葺	蕗	伏	副	復	幅	服		9590	4180
	4A20		福	腹	複	覆	淵	弗	払	沸	仏	物	鮒	分	吻	噴	墳	959E	4200
	4A30	憤	扮	焚	奮	粉	糞	紛	雰	文	聞							95AE	4216
へ												丙	併	兵	塀	幣	平		
	4A40	弊	柄	並	蔽	閉	陛	米	頁	僻	壁	癖	碧	別	瞥	蔑	箆	95BE	4232
	4A50	偏	変	片	篇	編	辺	返	遍	便	勉	娩	弁	鞭				95CE	4248
ほ															保	舗	鋪		
	4A60	圃	捕	歩	甫	補	輔	穂	募	墓	慕	戊	暮	母	簿	菩	倣	95DE	4264
	4A70	俸	包	呆	報	奉	宝	峰	峯	崩	庖	抱	捧	放	方	朋		95EE	4280
	4B20		法	泡	烹	砲	縫	胞	芳	萌	蓬	蜂	褒	訪	豊	邦	鋒	963F	4300
	4B30	飽	鳳	鵬	乏	亡	傍	剖	坊	妨	帽	忘	忙	房	暴	望	某	964F	4316
	4B40	棒	冒	紡	肪	膨	謀	貌	貿	鉾	防	吠	頬	北	僕	卜	墨	965F	4332
	4B50	撲	朴	牧	睦	穆	釦	勃	没	殆	堀	幌	奔	本	翻	凡	盆	966F	4348
ま	4B60	摩	磨	魔	麻	埋	妹	昧	枚	毎	哩	槙	幕	膜	枕	鮪	柾	9680	4364
	4B70	鱒	桝	亦	俣	又	抹	末	沫	迄	侭	繭	麿	万	慢	満		9690	4380
	4C20		漫	蔓														969E	4400
み					味	未	魅	巳	箕	岬	密	蜜	湊	蓑	稔	脈	妙		
	4C30	粍	民	眠	務													96AE	4416
む						夢	無	牟	矛	霧	鵡	椋	婿	娘					
め															冥	名	命		

	区 点	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		区 点
	JIS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	シフトJIS	
め	4C40	明	盟	迷	銘	鳴	姪	牝	滅	免	棉	綿	緬	面	麺			96BE	4432
																摸	模		
も	4C50	茂	妄	孟	毛	猛	盲	網	耗	蒙	儲	木	黙	目	杢	勿	餅	96CE	4448
	4C60	尤	戻	籾	貰	問	悶	紋	門	匁								96DE	4464
											也	冶	夜	爺	耶	野	弥		
や	4C70	矢	厄	役	約	薬	訳	躍	靖	柳	薮	鑓						96EE	4480
													愉	愈	油	癒			
ゆ	4D20		諭	輸	唯	佑	優	勇	友	宥	幽	悠	憂	揖	有	柚	湧	973F	4500
	4D30	涌	猶	猷	由	祐	裕	誘	遊	邑	郵	雄	融	夕				974F	4516
															予	余	与		
よ	4D40	誉	輿	預	傭	幼	妖	容	庸	揚	揺	擁	曜	楊	様	洋	溶	975F	4532
	4D50	熔	用	窯	羊	耀	葉	蓉	要	謡	踊	遥	陽	養	慾	抑	欲	976F	4548
	4D60	沃	浴	翌	翼	淀												9780	4564
							羅	螺	裸	来	莱	頼	雷	洛	絡	落	酪		
ら	4D70	乱	卵	嵐	欄	濫	藍	蘭	覧									9790	4580
										利	吏	履	李	梨	理	璃			
り	4E20		痢	裏	裡	里	離	陸	律	率	立	葎	掠	略	劉	流	溜	979E	4600
	4E30	琉	留	硫	粒	隆	竜	龍	侶	慮	旅	虜	了	亮	僚	両	凌	97AE	4616
	4E40	寮	料	梁	涼	猟	療	瞭	稜	糧	良	諒	遼	量	陵	領	力	97BE	4632
	4E50	緑	倫	厘	林	淋	燐	琳	臨	輪	隣	鱗	麟					97CE	4648
														瑠	塁	涙	累		
る	4E60	類																97DE	4664
			令	伶	例	冷	励	嶺	怜	玲	礼	苓	鈴	隷	零	霊	麗		
れ	4E70	齢	暦	歴	列	劣	烈	裂	廉	恋	憐	漣	煉	簾	練	聯		97EE	4680
	4F20		蓮	連	錬													983F	4700
						呂	魯	櫓	炉	賂	路	露	労	婁	廊	弄	朗		
ろ	4F30	楼	榔	浪	漏	牢	狼	篭	老	聾	蝋	郎	六	麓	禄	肋	録	984F	4716
	4F40	論																985F	4732
			倭	和	話	歪	賄	脇	惑	枠	鷲	亙	亘	鰐	詫	藁	蕨		
わ	4F50	椀	湾	碗	腕													986F	4748

		区点	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	シフトJIS	区点
		JIS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
一画	一	5020		弌	丐	丕													989E	4800
	丨						个	丱												
	丶								丶	丼										
	丿										丿	乂	乖	乘						
	乙														亂					
	亅															亅	豫	亊		
		5030	舒																98AE	4816
二画	二			弍	于	亞	亟													
	亠							亠	亢	京	亳	亶								
	人												从	仍	仄	仆	仂	仗	98BE	4832
		5040	仞	仭	仟	价	伉	佚	估	佛	佝	佗	佇	佶	侈	侏	侘	佻	98CE	4848
		5050	佩	佰	侑	佯	來	侖	儘	俔	俟	俎	俘	俛	俑	俚	俐	俤	98DE	4864
		5060	俥	倚	倨	倔	倪	倥	倅	伜	俶	倡	倩	倬	俾	俯	們	倆	98EE	4880
		5070	偃	假	會	偕	偐	偈	做	偖	偬	偸	傀	傚	傅	傴	傲		993F	4900
		5120		僉	僊	傳	僂	僖	僞	僥	僭	僣	僮	價	僵	儉	儁	儂	994F	4916
		5130	儖	儕	儔	儚	儡	儺	儷	儼	儻									
	儿											儿	兀	兒	兌	兔	兢	競		
	入	5140	兩	兪															995F	4932
	八				兮	冀														
	冂						冂	囘	册	冉	冏	冑	冓	冕						
	冖														冖	冤	冦	冢		
		5150	冩	冪															996F	4648
	冫				冫	决	冱	冲	冰	况	冽	凅	凉	凛						
	几														几	處	凩	凭		
		5160	凰																9980	4964
	凵			凵	凾															
	刀					刄	刋	刔	刎	刧	刪	刮	刳	刹	剏	剄	剋	剌		
		5170	剞	剔	剪	剴	剩	剳	剿	剽	劍	劔	劒	剱	劈	劑	辨		9990	4980
		5220		辧															999E	5000
	力				劬	劭	劼	劵	勁	勍	勗	勞	勣	勦	飭	勠	勳	勵		
		5230	勸																99AE	5016
	勹			勹	匆	匈	甸	匍	匐	匏										
	匕										匕									
	匚											匚	匣	匯	匱	匳				
	匸																匸	區		
	十	5240	卆	卅	丗	卉	卍	凖											99BE	5032
	卜								卞											
	卩									卩	卮	夘	卻	卷						
	厂														厂	厖	厠	厦		

		区 点	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	シフトJIS	区 点
		JIS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
二画	厂	5250	厥	厮	厰														99CE	5048
	厶					厶	參	篡												
	又								雙	叟	曼	燮								
三画	口												叮	叨	叭	叺	吁	吽		
		5260	呀	听	吭	吼	吮	吶	吩	吝	呎	咏	呵	咎	呟	呱	呷	呰	99DE	5064
		5270	咒	呻	咀	呶	咄	咐	咆	哇	咢	咸	咥	咬	哄	哈	咨		99EE	5080
		5320		咫	哂	咤	咾	咼	哘	哥	哦	唏	唔	哽	哮	哭	哺	哢	9A3F	5100
		5330	唹	啀	啣	啌	售	啜	啅	啖	啗	唸	唳	啝	喙	喀	咯	喊	9A4F	5116
		5340	喟	啻	啾	喘	喞	單	啼	喃	喩	喇	喨	嗚	嗅	嗟	嗄	嗜	9A5F	5132
		5350	嗤	嗔	嘔	嗷	嘖	嗾	嗽	嘛	嗹	噎	噐	營	嘴	嘶	嘲	嘸	9A6F	5148
		5360	噫	噤	嘯	噬	噪	嚆	嚀	嚊	嚠	嚔	嚏	嚥	嚮	嚶	嚴	囂	9A80	5164
		5370	嚼	囁	囃	囀	囈	囎	囑	囓									9A90	5480
	囗										囗	囮	囹	圀	囿	圄	圉			
		5420		圈	國	圍	圓	團	圖	嗇	圜								9A9E	5200
	土											圦	圷	圸	坎	圻	址	坏		
		5430	坩	埀	垈	坡	坿	垉	垓	垠	垳	垤	垪	垰	埃	埆	埔	埒	9AAE	5216
		5440	埓	堊	埖	埣	堋	堙	堝	塲	堡	塢	塋	塰	毀	塒	堽	塹	9ABE	5232
		5450	墅	墹	墟	墫	墺	壞	墻	墸	墮	壅	壓	壑	壗	壙	壘	壥	9ACE	5248
		5460	壜	壤	壟														9ADE	5264
	士					壯	壺	壹	壻	壼	壽									
	夂											夂	夊	夐						
	夕														夛	梦	夥			
	大																	夬		
		5470	夭	夲	夸	夾	竒	奕	奐	奎	奚	奘	奢	奠	奧	奬	奩		9AEE	5280
	女	5520		奸	妁	妝	佞	侫	妣	妲	姆	姨	姜	妍	姙	姚	娥	娟	9B3F	5300
		5530	娑	娜	娉	娚	婀	婬	婉	娵	娶	婢	婪	媚	媼	媾	嫋	嫂	9B4F	5316
		5540	媽	嫣	嫗	嫦	嫩	嫖	嫺	嫻	嬌	嬋	嬖	嬲	嫐	嬪	嬶	嬾	9B5F	5332
		5550	孃	孅	孀														9B6F	5348
	子					孑	孕	孚	孛	孥	孩	孰	孳	孵	學	斈	孺			
	宀																	宀		
		5560	它	宦	宸	寃	寇	寉	寔	寐	寤	實	寢	寞	寥	寫	寰	寶	9B80	5364
		5570	寳																9B90	5380
	寸			尅	將	專	對													
	小							尓	尠											
	尢									尢	尨									
	尸											尸	尹	屁	屆	屎	屓			
		5620		屐	屏	孱	屬												9B9E	5400
	屮							屮												
	山								乢	屶	屹	岌	岑	岔	妛	岫	岻	岶		

		区点	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		区点
		JIS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	シフトJIS	
	山	5630	岼	岷	峅	岾	峇	峙	峩	峽	峺	峭	嶌	峪	崋	崕	崗	嵜	9BAE	5416
		5640	崟	崛	崑	崔	崢	崚	崙	崘	嵌	嵒	嵎	嵋	嵬	嵳	嵶	嶇	9BBE	5432
		5650	嶄	嶂	嶢	嶝	嶬	嶮	嶽	嶐	嶷	嶼	巉	巍	巓	巒	巖		9BCE	5448
	巛																	巛		
	工	5660	巫																9BDE	5464
	己			已	巵															
	巾					帋	帚	帙	帑	帛	帶	帷	幄	幃	幀	幎	幗	幔		
		5670	幟	幢	幤	幇													9BEE	5480
三	干						幵	并												
画	幺								幺	麼										
	广										广	庠	廁	廂	廈	廐	廏			
		5720		廖	廣	廝	廚	廛	廢	廡	廨	廩	廬	廱	廳	廰			9C3F	5500
	廴																廴	廸		
	廾	5730	廾	弃	弉	彝	彜												9C4F	5516
	弋							弋	弑											
	弓									弖	弩	弭	弸	彁	彈	彌	彎	弯		
	彑	5740	彑	彖	彗	彙													9C5F	5532
	彡						彡	彭												
	彳								彳	彷	徃	徂	彿	徊	很	徑	徇	從		
		5750	徙	徘	徠	徨	徭	徼											9C6F	5548
	心								忖	忻	忤	忸	忱	忝	悳	忿	怡	恠		
		5760	怙	怐	怩	怎	怱	怛	怕	怫	怦	怏	怺	恚	恁	恪	恷	恟	9C80	5564
		5770	恊	恆	恍	恣	恃	恤	恂	恬	恫	恙	悁	悍	惧	悃	悚		9C90	5580
		5820		悄	悛	悖	悗	悒	悧	悋	惡	悸	惠	惓	悴	忰	悽	惆	9C9E	5600
		5830	悵	惘	慍	愕	愆	惶	惷	愀	惴	惺	愃	愡	惻	惱	愍	愎	9CAE	5616
		5840	慇	愾	愨	愧	慊	愿	愼	愬	愴	愽	慂	慄	慳	慷	慘	慙	9CBE	5632
		5850	慚	慫	慴	慯	慥	慱	慟	慝	慓	慵	憙	憖	憇	憬	憔	憚	9CCE	5648
		5860	憊	憑	憫	憮	懌	懊	應	懷	懈	懃	懆	憺	懋	罹	懍	懦	9CDE	5664
四		5870	懣	懶	懺	懴	懿	懽	懼	懾	戀								9CEE	5680
画	戈											戈	戉	戍	戌	戔	戛			
		5920		戞	戡	截	戮	戰	戲	戳									9D3F	5700
	戸										扁	扎	扞	扣	扛	扠	扨	扼		
	手	5930	抂	抉	找	抒	抓	抖	拔	抃	抔	拗	拑	抻	拏	拿	拆	擔	9D4F	5716
		5940	拈	拜	拌	拊	拂	拇	抛	拉	挌	拮	拱	挧	挂	挈	拯	拵	9D5F	5732
		5950	捐	挾	捍	搜	捏	掖	掎	掀	掫	捶	掣	掏	掉	掟	掵	捫	9D6F	5748
		5960	捩	掾	揩	揀	揆	揣	揉	插	揶	揄	搖	搴	搆	搓	搦	搶	9D80	5764
		5970	攝	搗	搨	搏	摧	摯	摶	摎	攪	撕	撓	撥	撩	撈	撼		9D90	5780
		5A20		據	擒	擅	擇	撻	擘	擂	擱	擧	舉	擠	擡	抬	擣	擯	9D9E	5800
		5A30	攬	擶	擴	擲	擺	攀	擽	攘	攜	攅	攤	攣	攫				9DAE	5816
	攴															攴	攵	攷		

		区点	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		区点
		JIS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	シフトJIS	
四画	攴	5A40	收	攸	畋	效	敖	敕	敍	敘	敞	敝	敲	數	斂	斃	變		9DBE	5832
	斗																	斛		
		5A50	斟																9DCD	5848
	斤			斫																
					斷															
	方					旃	旆	旁	旄	旌	旒	旛	旙							
	无													无	旡					
	日															旱	杲	昊		
		5A60	昃	旻	杳	昵	昶	昴	昜	晏	晄	晉	晁	晞	晝	晤	晧	晨	9DDE	5864
		5A70	晟	晢	晰	暃	暈	暎	暉	暄	暘	暝	曁	暹	曉	暾	暼		9DEE	5880
		5B20		曄	暸	曖	曚	曠	昿	曦	曩								9E3F	5900
	曰											曰	曵	曷						
	月														朏	朖	朞	朦		
		5B30	朧	霸															9E4F	5916
	木				朮	朿	朶	杁	朸	朷	杆	杞	杠	杙	杣	杤	枉	杰		
		5B40	枩	杼	杪	枌	枋	枦	枡	枅	枷	柯	枴	柬	枳	柩	枸	柤	9E5F	5932
		5B50	柞	柝	柢	柮	枹	柎	柆	柧	檜	栞	框	栩	桀	桍	栲	桎	9E6F	5948
		5B60	梳	栫	桙	档	桷	桿	梟	梏	梭	梔	條	梛	梃	檮	梹	桴	9E80	5964
		5B70	梵	梠	梺	椏	梍	桾	椁	棊	椈	棘	椢	椦	棡	椌	棍		9E90	5980
		5C20		棔	棧	棕	椶	椒	椄	棗	棣	椥	棹	棠	棯	椨	椪	椚	9E9E	6000
		5C30	椣	椡	棆	楹	楷	楜	楸	楫	楔	楾	楮	椹	楴	椽	楙	椰	9EAE	6016
		5C40	楡	楞	楝	榁	楪	榲	榮	槐	榿	槁	槓	榾	槎	寨	槊	槝	9EBE	6032
		5C50	榻	槃	榧	樮	榑	榠	榜	榕	榴	槞	槨	樂	樛	槿	權	槹	9ECE	6048
		5C60	槲	槧	樅	榱	樞	槭	樔	槫	樊	樒	櫁	樣	樓	橄	樌	橲	9EDE	6064
		5C70	樶	橸	橇	橢	橙	橦	橈	樸	樢	檐	檍	檠	檄	檢	檣		9EEE	6080
		5D20		檗	蘗	檻	櫃	櫂	檸	檳	檬	櫞	櫑	櫟	檪	櫚	櫪	櫻	9F3F	6100
		5D30	欅	蘖	櫺	欒	欖	鬱	欟										9F4F	6116
	欠									欸	欷	盜	欹	飮	歇	歃	歉	歐		
		5D40	歙	歔	歛	歟	歡												9F5F	6132
	止							歸												
	歹								歹	歿	殀	殄	殃	殍	殘	殕	殞	殤		
		5D50	殪	殫	殯	殲	殱												9F6F	6148
	殳							殳	殷	殼	毆									
	毋											毋	毓							
	毛													毟	毬	毫	毳	毯		
		5D60	麾	氈															5F80	6164
	氏				氓															
	气					气	氛	氤	氣											
	水									汞	汕	汢	汪	沂	沍	沚	沁	沛		

画数	部首	区 点	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		区 点
		JIS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	シフトJIS	
四画	水	5D70	汾	汨	汳	沒	沐	泄	泱	泓	沽	泗	泅	泝	沮	沱	沾		9F90	6180
		5E20		沺	泛	泯	泙	泪	洟	衍	洶	洫	洽	洸	洙	洵	洳	洒	9F9E	6200
		5E30	洌	浣	涓	浤	浚	浹	浙	涎	涕	濤	涅	淹	渕	渊	涵	淇	9FAE	6216
		5E40	淦	涸	淆	淬	淞	淌	淨	淒	淅	淺	淙	淤	淕	淪	淮	渭	9FBE	6232
		5E50	湮	渮	渙	湲	湟	渾	渣	湫	渫	湶	湍	渟	湃	渺	湎	渤	9FCE	6248
		5E60	滿	渝	游	溂	溪	溘	滉	溷	滓	溽	溯	滄	溲	滔	滕	溏	9FDE	6264
		5E70	溥	滂	溟	潁	漑	灌	滬	滸	滾	漿	滲	漱	滯	漲	滌		9FEE	6280
		5F20		漾	漓	滷	澆	潺	潸	澁	澀	潯	潛	濳	潭	澂	潼	潘	E03F	6300
		5F30	澎	澑	濂	潦	澳	澣	澡	澤	澹	濆	澪	濟	濕	濬	濔	濘	E04F	6316
		5F40	濱	濮	濛	瀉	瀋	濺	瀑	瀁	瀏	濾	瀛	瀚	潴	瀝	瀘	瀟	E05F	6332
		5F50	瀰	瀾	瀲	灑	灣												E06F	6348
	火							炙	炒	炯	烱	炬	炸	炳	炮	烟	烋	烝		
		5F60	烙	焉	烽	焜	焙	煥	煕	熈	煦	煢	煌	煖	煬	熏	燻	熄	E080	6364
		5F70	熕	熨	熬	燗	熹	熾	燒	燉	燔	燎	燠	燬	燧	燵	燼		E090	6380
		6020		燹	燿	爍	爐	爛	爨										E09E	6400
	爪									爭	爬	爰	爲							
	爻													爻	爼					
	爿															爿	牀	牆		
		6030	牋	牘															E0AE	6416
	牛				牴	牾	犂	犁	犇	犒	犖	犢	犧							
	犬													犹	犲	狃	狆	狄		
		6040	狎	狒	狢	狠	狡	狹	狷	倏	猗	猊	猜	猖	猝	猴	猯	猩	E0BE	6432
		6050	猥	猾	獎	獏	默	獗	獪	獨	獰	獸	獵	獻	獺				E0CE	6448
	王															珈	玳	珎		
		6060	玻	珀	珥	珮	珞	璢	琅	瑯	琥	珸	琲	琺	瑕	琿	瑟	瑙	E0DE	6464
		6070	瑁	瑜	瑩	瑰	瑣	瑪	瑶	瑾	璋	璞	璧	瓊	瓏	瓔	珱		E0EE	6480
五画	瓜	6120		瓠	瓣														E13F	6500
	瓦					瓧	瓩	瓮	瓲	瓰	瓱	瓸	瓷	甄	甃	甅	甌	甎		
		6130	甍	甕	甓														E14F	6516
	甘					甞														
	生						甦													
	用							甬												
	田								甼	畄	畍	畊	畉	畛	畆	畚	畩	畤		
		6140	畧	畫	畭	畸	當	疆	疇	畴	疊	疉	疂						E15F	6532
	疒													疔	疚	疝	疥	疣		
		6150	痂	疳	痃	疵	疽	疸	疼	疱	痍	痊	痒	痙	痣	痞	痾	痿	E16F	6548
		6160	痼	瘁	痰	痺	痲	痳	瘋	瘍	瘉	瘟	瘧	瘠	瘡	瘢	瘤	瘴	E180	6564
		6170	瘰	瘻	癇	癈	癆	癜	癘	癡	癢	癨	癩	癪	癧	癬	癰		E190	6580

画数	部首	区点	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	シフトJIS	区点
		JIS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
五画	疒	6220		癲															E19E	6600
	癶				癶	癸	發													
	白							皀	皃	皈	皋	皎	皖	皓	皙	皚				
	皮																皰	皴		
		6230	皸	皹	皺														E1AE	6616
	皿					盂	盍	盖	盒	盞	盡	盥	盧	盪	蘯					
	目															盻	眈	眇		
		6240	眄	眩	眤	眞	眥	眦	眛	眷	眸	睇	睚	睨	睫	睛	睥	睿	E1BE	6632
		6250	睾	睹	瞎	瞋	瞑	瞠	瞞	瞰	瞶	瞹	瞿	瞼	瞽	瞻	矇	矍	E1CE	6648
		6260	矗	矚															E1DE	6664
	矛				矜															
	矢					矣	矮													
	石							矼	砌	砒	礦	砠	礪	硅	碎	硴	碆	硼		
		6270	碚	碌	碣	碵	碪	碯	磑	磆	磋	磔	碾	碼	磅	磊	磬		E1EE	6680
		6320		磧	磚	磽	磴	礇	礒	礑	礙	礬	礫						E23F	6700
	示													祀	祠	祗	祟	祚		
		6330	祕	祓	祺	祿	禊	禝	禧	齋	禪	禮	禳						E24F	6716
	禸													禹	禺					
	禾															秉	秕	秧		
		6340	秬	秡	秣	稈	稍	稘	稙	稠	稟	禀	稱	稻	稾	稷	穃	穗	E25F	6732
		6350	穉	穡	穢	穩	龝	穰											E26F	6748
	穴								穹	穽	窈	窗	窕	窘	窖	窩	竈	窰		
		6360	窶	竅	竄	窿	邃	竇	竊										E280	6764
	立									竍	竏	竕	竓	站	竚	竝	竡	竢		
		6370	竦	竭	竰	笂	笏	笊	笆	笳	笘	笙	笞	笵	笨	笶	筐		E290	6780
六画	竹	6420		筺	笄	筍	笋	筌	筅	筵	筥	筴	筧	筰	筱	筬	筮	箝	E29E	6800
		6430	箘	箟	箍	箜	箚	箋	箒	箏	筝	箙	篋	篁	篌	篏	箴	篆	E2AE	6816
		6440	篝	篩	簑	簔	篦	篥	籠	簀	簇	簓	篳	篷	簗	簍	篶	簣	E2BE	6832
		6450	簧	簪	簟	簷	簫	簽	籌	籃	籔	籏	籀	籐	籘	籟	籤	籖	E2CE	6848
		6460	籥	籬															E2DE	6864
	米				籵	粃	粐	粤	粭	粢	粫	粡	粨	粳	粲	粱	粮	粹		
		6470	粽	糀	糅	糂	糘	糒	糜	糢	鬻	糯	糲	糴	糶				E2EE	6880
	糸															糺	紆			
		6520		紂	紜	紕	紊	絅	絋	紮	紲	紿	紵	絆	絳	絖	絎	絲	E33F	6900
		6530	絨	絮	絏	絣	經	綉	絛	綏	絽	綛	綺	綮	綣	綵	緇	綽	E34F	6916
		6540	綫	總	綢	綯	緜	綸	綟	綰	緘	緝	緤	緞	緻	緲	緡	縅	E35F	6932
		6550	縊	縣	縡	縒	縱	縟	縉	縋	縢	繆	繦	縻	縵	縹	繃	縷	E36F	6948
		6560	縲	縺	繧	繝	繖	繞	繙	繚	繹	繪	繩	繼	繻	纃	緕	繽	E380	6964
		6570	辮	繿	纈	纉	續	纒	纐	纓	纔	纖	纎	纛	纜				E390	6980
	缶															缸	缺			

画数	部首	区点 / JIS	0 / 0	1 / 1	2 / 2	3 / 3	4 / 4	5 / 5	6 / 6	7 / 7	8 / 8	9 / 9	10 / A	11 / B	12 / C	13 / D	14 / E	15 / F	シフトJIS	区点
六画	缶	6620		罅	罌	罍	罎	罐											E39E	7000
	网								网	罕	罔	罘	罟	罠	罨	罩	罧	罸		
		6630	羂	羆	羃	羈	羇												E3AE	7016
	羊							羌	羔	羞	羝	羚	羣	羯	羲	羹	羮	羶		
		6640	羸	譱															E3BE	7032
	羽				翅	翆	翊	翕	翔	翡	翦	翩	翳	翹	飜					
	老															耆	耄	耋		
	耒	6650	耒	耘	耙	耜	耡	耨											E3CE	7048
	耳								耿	耻	聊	聆	聒	聘	聚	聟	聢	聨		
		6660	聳	聲	聰	聶	聹	聽											E3DE	7064
	聿								聿	肄	肆	肅								
	肉												肛	肓	肚	肭	冐	肬		
		6670	胛	胥	胙	胝	胄	胚	胖	脉	胯	胱	脛	脩	脣	脯	腋		E3EE	7080
		6720		隋	腆	脾	腓	腑	胼	腱	腮	腥	腦	腴	膃	膈	膊	膀	E43F	7100
		6730	膂	膠	膕	膤	膣	腟	膓	膩	膰	膵	膾	膸	膽	臀	臂	膺	E44F	7116
		6740	臉	臍	臑	臙	臘	臈	臚	臟	臠								E45F	7132
	臣											臧								
	至												臺	臻						
	臼														臾	舁	舂	舅		
		6750	與	舊															E46F	7148
	舌				舍	舐	舖													
	舟							舩	舫	舸	舳	艀	艙	艘	艝	艚	艟	艤		
		6760	艢	艨	艪	艫	舮												E480	7164
	艮							艱												
	色								艷											
	艸									艸	艾	芍	芒	芫	芟	芻	芬	苡		
		6770	苣	苟	苒	苴	苳	苺	莓	范	苻	苹	苞	茆	苜	茉	苙		E490	7180
		6820		茵	茴	茖	茲	茱	荀	茹	荐	荅	茯	茫	茗	茘	莅	莚	E49E	7200
		6830	莪	莟	莢	莖	茣	莎	莇	莊	荼	莵	荳	荵	莠	莉	莨	菴	E4AE	7216
		6840	萓	菫	菎	菽	萃	菘	萋	菁	菷	萇	菠	菲	萍	萢	萠	莽	E4BE	7232
		6850	萸	蔆	菻	葭	萪	萼	蕚	蒄	葷	葫	蒭	葮	蒂	葩	葆	萬	E4CE	7248
		6860	葯	葹	萵	蓊	葢	蒹	蒿	蒟	蓙	蓍	蒻	蓚	蓐	蓁	蓆	蓖	E4DE	7264
		6870	蒡	蔡	蓿	蓴	蔗	蔘	蔬	蔟	蔕	蔔	蓼	蕀	蕣	蕘	蕈		E4EE	7280
		6920		蕁	蘂	蕋	蕕	薀	薤	薈	薑	薊	薨	蕭	薔	薛	藪	薇	E53F	7300
		6930	薜	蕷	蕾	薐	藉	薺	藏	薹	藐	藕	藝	藥	藜	藹	蘊	蘓	E54F	7316
		6940	蘋	藾	藺	蘆	蘢	蘚	蘰	蘿									E55F	7332
	虍										虍	乕	虔	號	虧					
	虫															虱	蚓	蚣		
		6950	蚩	蚪	蚋	蚌	蚶	蚯	蛄	蛆	蚰	蛉	蠣	蚫	蛔	蛞	蛩	蛬	E56F	7348
		6960	蛟	蛛	蛯	蜒	蜆	蜈	蜀	蜃	蛻	蜑	蜉	蜍	蛹	蜊	蜴	蜿	E580	7364

画数	部首	区点	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		区点
		JIS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	シフトJIS	
六画	虫	6970	蜷	蜻	蜥	蜩	蜚	蝠	蝟	蝸	蝌	蝎	蝴	蝗	蝨	蝮	蝙		E590	7380
		6A20		蝓	蝣	蝪	蠅	螢	螟	螂	螯	蟋	螽	蟀	蟐	雖	螫	蟄	E59E	7400
		6A30	螳	蟇	蟆	螻	蟯	蟲	蟠	蠏	蠍	蟾	蟶	蟷	蠎	蟒	蠑	蠖	E5AE	7416
		6A40	蠕	蠢	蠡	蠱	蠶	蠹	蠧	蠻									E5BE	7432
	血										衄	衂								
	行												衒	衙	衞	衢				
	衣																衫	袁		
		6A50	衾	袞	衵	衽	袵	衲	袂	袗	袒	袮	袙	袢	袍	袤	袰	袿	E5CE	7448
		6A60	袱	裃	裄	裔	裘	裙	裝	裹	褂	裼	裴	裨	裲	褄	褌	褊	E5DE	7464
		6A70	褓	襃	褞	褥	褪	褫	襁	襄	褻	褶	褸	襌	褝	襠	襞		E5EE	7480
		6B20		襦	襤	襭	襪	襯	襴	襷									E63F	7500
七画	襾										襾	覃	覈	覊						
	見														覓	覘	覡	覩		
		6B30	覦	覬	覯	覲	覺	覽	覿	觀									E64F	7516
	角										觚	觜	觝	觧	觴	觸				
	言																訃	訖		
		6B40	訐	訌	訛	訝	訥	訶	詁	詛	詒	詆	詈	詼	詭	詬	詢	誅	E65F	7532
		6B50	誂	誄	誨	誡	誑	誥	誦	誚	誣	諄	諍	諂	諚	諫	諳	諧	E66F	7548
		6B60	諤	諱	謔	諠	諢	諷	諞	諛	謌	謇	謚	諡	謖	謐	謗	謠	E680	7564
		6B70	謳	鞫	謦	謫	謾	謨	譁	譌	譏	譎	證	譖	譛	譚	譫		E690	7580
		6C20		譟	譬	譯	譴	譽	讀	讌	讎	讒	讓	讖	讙	讚			E69E	7600
	谷																谺	豁		
		6C30	谿																E6AE	7616
	豆			豈	豌	豎	豐													
	豕							豕	豢	豬										
	豸										豸	豺	貂	貉	貅	貊	貍	貎		
		6C40	貔	豼	貘														E6BE	7632
	貝					戝	貭	貪	貽	貲	貳	貮	貶	賈	賁	賤	賣	賚		
		6C50	賽	賺	賻	贄	贅	贊	贇	贏	贍	贐	齎	贓	賍	贔	贖		E6CE	7648
	赤																	赧		
		6C60	赭																E6DE	7664
	走			赱	赳	趁	趙													
	足							跂	趾	趺	跏	跚	跖	跌	跛	跋	跪	跫		
		6C70	跟	跣	跼	踈	踉	跿	踝	踞	踐	踟	蹂	踵	踰	踴	蹊		E6EE	7680
		6D20		蹇	蹉	蹌	蹐	蹈	蹙	蹤	蹠	踪	蹣	蹕	蹶	蹲	蹼	躁	E73F	7700
		6D30	躇	躅	躄	躋	躊	躓	躑	躔	躙	躪	躡						E74F	7716
	身													躬	躰	軆	躱	躾		
		6D40	軅	軈															E75F	7732
	車				軋	軛	軣	軼	軻	軫	軾	輊	輅	輕	輒	輙	輓	輜		
		6D50	輟	輛	輌	輦	輳	輻	輹	轅	轂	輾	轌	轉	轆	轎	轗	轜	E76F	7748

画数	部首	区点 / JIS	0 / 0	1 / 1	2 / 2	3 / 3	4 / 4	5 / 5	6 / 6	7 / 7	8 / 8	9 / 9	10 / A	11 / B	12 / C	13 / D	14 / E	15 / F	シフトJIS	区点
七画	車	6D60	轢	轣	轤														E780	7764
	辛					辜	辟	辣	辭	辯										
	辵										辷	迚	迥	迢	迪	迯	邇	迴		
		6D70	逅	迹	迺	逑	逕	逡	逍	逞	逖	逋	逧	逶	逵	逹	迸		E790	7780
		6E20		遏	遐	遑	遒	逎	遉	逾	遖	遘	遞	遨	遯	遶	隨	遲	E79E	7800
		6E30	邂	遽	邁	邀	邊	邉	邏										E7AE	7816
	邑									邨	邯	邱	邵	郢	郤	扈	郛	鄂		
		6E40	鄒	鄙	鄲	鄰													E7BE	7832
	酉						酊	酖	酘	酣	酥	酩	酳	酲	醋	醉	醂	醢		
		6E50	醫	醯	醪	醵	醴	醺	釀	釁									E7CE	7848
	釆										釉	釋								
	里												釐							
八画	金													釖	釟	釡	釛	釼		
		6E60	釵	釶	鈞	釿	鈔	鈬	鈕	鈑	鉞	鉗	鉅	鉉	鉤	鉈	銕	鈿	E7DE	7864
		6E70	鉋	鉐	銜	銖	銓	銛	鉚	鋏	銹	銷	鋩	錏	鋺	鍄	錮		E7EE	7880
		6F20		錙	錢	錚	錣	錺	錵	錻	鍜	鍠	鍼	鍮	鍖	鎰	鎬	鎭	E83F	7900
		6F30	鎔	鎹	鏖	鏗	鏨	鏥	鏘	鏃	鏝	鏐	鏈	鏤	鐚	鐔	鐓	鐃	E84F	7916
		6F40	鐇	鐐	鐶	鐫	鐵	鐡	鐺	鑁	鑒	鑄	鑛	鑠	鑢	鑞	鑪	鈩	E85F	7932
		6F50	鑰	鑵	鑷	鑽	鑚	鑼	鑾	钁	鑿								E86F	7948
	門											閂	閇	閊	閔	閖	閘	閙		
		6F60	閠	閨	閧	閭	閼	閻	閹	閾	闊	濶	闃	闍	闌	闕	闔	闖	E880	7964
		6F70	關	闡	闥	闢													E890	7980
	阜						阡	阨	阮	阯	陂	陌	陏	陋	陷	陜	陞			
		7020		陝	陟	陦	陲	陬	隍	隘	隕	隗	險	隧	隱	隲	隰	隴	E89E	8000
	隶	7030	隶	隸															E8AE	8016
	隹				隹	雎	雋	雉	雍	襍	雜	霍	雕							
	雨													雹	霄	霆	霈	霓		
		7040	霎	霑	霏	霖	霙	霤	霪	霰	霹	霽	霾	靄	靆	靈	靂	靉	E8BE	8032
	青	7050	靜																E8CE	8048
	非			靠																
九画	面				靤	靦	靨													
	革							勒	靫	靱	靹	鞅	靼	鞁	靺	鞆	鞋	鞏		
		7060	鞐	鞜	鞨	鞦	鞣	鞳	鞴	韃	韆	韈							E8DE	8064
	韋												韋	韜						
	韭														韭	齏	韲			
	音																	竟		
		7070	韶	韵															E8EE	8080
	頁				頏	頌	頸	頤	頡	頷	頽	顆	顏	顋	顫	顯	顰			

		区点	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		区点
		JIS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	シフトJIS	
九画	頁	7120		顱	顴	顳													E93F	8100
	風						颪	颯	颱	颶	飄	飃	飆							
	食													飩	飫	餃	餉	餒		
		7130	餔	餘	餡	餝	餞	餤	餠	餬	餮	餽	餾	饂	饉	饅	饐	饋	E94F	8116
		7140	饑	饒	饌	饕													E95F	8132
	首						馗	馘												
	香								馥											
十画	馬									馭	馮	馼	駟	駛	駝	駘	駑	駭		
		7150	駮	駱	駲	駻	駸	騁	騏	騅	駢	騙	騫	騷	驅	驂	驀	驃	E96F	8148
		7160	騾	驕	驍	驛	驗	驟	驢	驥	驤	驩	驫	驪					E980	8164
	骨														骭	骰	骼	髀		
		7170	髏	髑	髓	體													E990	8180
	高						髞													
	髟							髟	髢	髣	髦	髯	髫	髮	髴	髱	髷			
		7220		髻	鬆	鬘	鬚	鬟	鬢	鬣									E99E	8200
	鬥										鬥	鬧	鬨	鬩	鬪	鬮				
	鬯																鬯			
	鬲																	鬲		
	鬼	7230	魄	魃	魏	魍	魎	魑	魘										E9AE	8216
十一画	魚									魴	鮓	鮃	鮑	鮖	鮗	鮟	鮠	鮨		
		7240	鮴	鯀	鯊	鮹	鯆	鯏	鯑	鯒	鯣	鯢	鯤	鯔	鯡	鰺	鯲	鯱	E9BE	8232
		7250	鯰	鰕	鰔	鰉	鰓	鰌	鰆	鰈	鰒	鰊	鰄	鰮	鰛	鰥	鰤	鰡	E9CE	8248
		7260	鰰	鱇	鰲	鱆	鰾	鱚	鱠	鱧	鱶	鱸							E9DE	8264
	鳥												鳧	鳬	鳰	鴉	鴈	鳫		
		7270	鴃	鴆	鴪	鴦	鶯	鴣	鴟	鵄	鴕	鴒	鵁	鴿	鴾	鵆	鵈		E9EE	8280
		7320		鵝	鵞	鵤	鵑	鵐	鵙	鵲	鶉	鶇	鶫	鵯	鵺	鶚	鶤	鶩	EA3F	8300
		7330	鶲	鷄	鷁	鶻	鶸	鶺	鷆	鷏	鷂	鷙	鷓	鷸	鷦	鷭	鷯	鷽	EA4F	8316
		7340	鸚	鸛	鸞														EA5F	8332
	鹵					鹵	鹹	鹽												
	鹿								麁	麈	麋	麌	麒	麕	麑	麝				
	麦																麥	麩		
		7350	麸	麪	麭														EA6F	8348
	麻					靡														
	黄						黌													
十二画	黍							黎	黏	黐										
	黒										黔	黜	點	黝	黠	黥	黨	黯		

		区 点	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		区 点
		JIS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	シフトJIS	
十二画以上	黒	7360	黴	黶	黷														EA80	8364
	黹					黹	黻	黼												
	黽								黽	鼇	鼈									
	鼓											皷	鼕							
	鼠													鼡	鼬					
	鼻															鼾				
	齊																齊			
	齒																	齒		
		7370	齔	齣	齟	齠	齡	齦	齧	齬	齪	齷	齲	齶					EA90	8380
	龍														龕					
	龜															龜				
	龠																龠			
		7420		堯	槇	遙	瑤												EA9E	8400
拡張漢字		7920		纊	褜	鍈	銈	蓜	俉	炻	昱	棈	鋹	曻	彅	丨	仡	仼	ED3F	8900
		7930	伀	伃	伹	佖	侒	侊	侚	侔	俍	偀	倢	俿	倞	偆	偰	偂	ED4F	8916
		7940	傔	僴	僘	兊	兤	冝	冾	凬	刕	劜	劦	勀	勛	匀	匇	匤	ED5F	8932
		7950	卲	厓	厲	叝	﨎	咜	咊	咩	哿	喆	坙	坥	垬	埈	埇	﨏	ED6F	8948
		7960	塚	增	墲	夋	奓	奛	奝	奣	妤	妺	孖	寀	甯	寘	寬	尞	ED80	8964
		7970	岦	岺	峵	崧	嵓	﨑	嵂	嵭	嶸	嶹	巐	弡	弴	彧	德		ED90	8980
		7A20		忞	恝	悅	悊	惞	惕	愠	惲	愑	愷	愰	憘	戓	抦	揵	ED9E	9000
		7A30	摠	撝	擎	敎	昀	昕	昻	昉	昮	昞	昤	晥	晗	晙	晴	晳	EDAE	9016
		7A40	暙	暠	暲	暿	曺	朎	朗	杦	枻	桒	柀	栁	桄	棏	﨓	楨	EDBE	9032
		7A50	﨔	榘	槢	樰	橫	橆	橳	橾	櫢	櫤	毖	氿	汜	沆	汯	泚	EDCE	9048
		7A60	洄	涇	浯	涖	涬	淏	淸	淲	淼	渹	湜	渧	渼	溿	澈	澵	EDDE	9064
		7A70	濵	瀅	瀇	瀨	炅	炫	焏	焄	煜	煆	煇	凞	燁	燾	犱		EDEE	9080
		7B20		犾	猤	猪	獷	玽	珉	珖	珣	珒	琇	珵	琦	琪	琩	琮	EE3F	9100
		7B30	瑢	璉	璟	甁	畯	皂	皜	皞	皛	皦	益	睆	劯	砡	硎	硤	EE4F	9116
		7B40	硺	礰	礼	神	祥	禔	福	禛	竑	竧	靖	竫	箞	精	絈	絜	EE5F	9132
		7B50	綷	綠	緖	繒	罇	羡	羽	茁	荢	荿	菇	菶	葈	蒴	蕓	蕙	EE6F	9148
		7B60	蕫	﨟	薰	蘒	﨡	蠇	裵	訒	訷	詹	誧	誾	諟	諸	諶	譓	EE80	9164
		7B70	譿	賰	賴	贒	赶	﨣	軏	﨤	逸	遧	郞	都	鄕	鄧	釚		EE90	9180
		7C20		釗	釞	釭	釮	釤	釥	鈆	鈐	鈊	鈺	鉀	鈼	鉎	鉙	鉑	EE9E	9200
		7C30	鈹	鉧	銧	鉷	鉸	鋧	鋗	鋙	鋐	﨧	鋕	鋠	鋓	錥	錡	鋻	EEAE	9216
		7C40	﨨	錞	鋿	錝	錂	鍰	鍗	鎤	鏆	鏞	鏸	鐱	鑅	鑈	閒	隆	EEBE	9232
		7C50	﨩	隝	隯	霳	霻	靃	靍	靏	靑	靕	顗	顥	飯	飼	餧	館	EECE	9248
		7C60	馞	驎	髙	髜	魵	魲	鮏	鮱	鮻	鰀	鵰	鵫	鶴	鸙	黑		EEDE	9264
		7C70		ⅰ	ⅱ	ⅲ	ⅳ	ⅴ	ⅵ	ⅶ	ⅷ	ⅸ	ⅹ	￢	￤	＇	＂		EEEE	9280

§4–11 キャラクターコード表

上位 / 下位	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		DLE (16)	ｽﾍﾟｰｽ (32)	0 (48)	@ (64)	P (80)	’ (96)	p (112)	▁ (128)	┴ (144)	ｽﾍﾟｰｽ (160)	ー (176)	タ (192)	ミ (208)	═ (224)	╳ (240)
1	SOH (1)	DC1 (17)	! (33)	1 (49)	A (65)	Q (81)	a (97)	q (113)	▂ (129)	┬ (145)	。 (161)	ア (177)	チ (193)	ム (209)	╞ (225)	円 (241)
2	STX (2)	DC2 (18)	” (34)	2 (50)	B (66)	R (82)	b (98)	r (114)	▃ (130)	┤ (146)	「 (162)	イ (178)	ツ (194)	メ (210)	╪ (226)	年 (242)
3	ETX (3)	DC3 (19)	# (35)	3 (51)	C (67)	S (83)	c (99)	s (115)	▄ (131)	├ (147)	」 (163)	ウ (179)	テ (195)	モ (211)	╡ (227)	月 (243)
4	EOT (4)	DC4 (20)	$ (36)	4 (52)	D (68)	T (84)	d (100)	t (116)	▅ (132)	▔ (148)	、 (164)	エ (180)	ト (196)	ヤ (212)	◢ (228)	日 (244)
5	ENQ (5)	NAK (21)	% (37)	5 (53)	E (69)	U (85)	e (101)	u (117)	▆ (133)	─ (149)	・ (165)	オ (181)	ナ (197)	ユ (213)	◣ (229)	時 (245)
6	ACK (6)	SYN (22)	& (38)	6 (54)	F (70)	V (86)	f (102)	v (118)	▇ (134)	│ (150)	ヲ (166)	カ (182)	ニ (198)	ヨ (214)	◥ (230)	分 (246)
7	BEL (7)	ETB (23)	’ (39)	7 (55)	G (71)	W (87)	g (103)	w (119)	█ (135)	▕ (151)	ァ (167)	キ (183)	ヌ (199)	ラ (215)	◤ (231)	秒 (247)
8	BS (8)	CAN (24)	((40)	8 (56)	H (72)	X (88)	h (104)	x (120)	▏ (136)	┌ (152)	ィ (168)	ク (184)	ネ (200)	リ (216)	♠ (232)	(248)
9	HT (9)	EM (25)	) (41)	9 (57)	I (73)	Y (89)	i (105)	y (121)	▎ (137)	┐ (153)	ゥ (169)	ケ (185)	ノ (201)	ル (217)	♥ (233)	(249)
A	LF (10)	SUB (26)	* (42)	: (58)	J (74)	Z (90)	j (106)	z (122)	▍ (138)	└ (154)	ェ (170)	コ (186)	ハ (202)	レ (218)	♦ (234)	(250)
B	VT (11)	ESC (27)	+ (43)	; (59)	K (75)	[(91)	k (107)	{ (123)	▌ (139)	┘ (155)	ォ (171)	サ (187)	ヒ (203)	ロ (219)	♣ (235)	(251)
C	FF (12)	→ (28)	, (44)	< (60)	L (76)	¥ (92)	l (108)	| (124)	▋ (140)	╭ (156)	ャ (172)	シ (188)	フ (204)	ワ (220)	● (236)	╲ (252)
D	CR (13)	← (29)	− (45)	= (61)	M (77)	] (93)	m (109)	} (125)	▊ (141)	╮ (157)	ュ (173)	ス (189)	ヘ (205)	ン (221)	○ (237)	(253)
E	SO (14)	↑ (30)	. (46)	> (62)	N (78)	^ (94)	n (110)	~ (126)	▉ (142)	╰ (158)	ョ (174)	セ (190)	ホ (206)	゛ (222)	╱ (238)	(254)
F	SI (15)	↓ (31)	/ (47)	? (63)	O (79)	_ (95)	o (111)	(127)	┼ (143)	╯ (159)	ッ (175)	ソ (191)	マ (207)	゜ (223)	╲ (239)	(255)

参考文献

- AIWA，B98-01 取扱説明書，AIWA
- C.F.Computing，DOS/V プログラマーズハンドブック，ソフトバンク
- CQ 出版，MS-DOS 基本プログラミング第 2 集 PC-9801 の割り込みと BIOS 活用法，CQ 出版社
- NEC，PC-9801VX ハードウェア マニュアル，NEC
- NEC，PC-9801DX ハードウェア マニュアル 他，NEC
- NEC，MS-DOS Ver5.0 拡張セット マニュアル（各種），NEC
- NEC，PC-8801Mk2FR N88-BASIC，N88-日本語 BASIC ガイドブック，NEC
- EPSON，PC286VG ユーザーズ マニュアル，EPSON
- EPSON， VP-130K 取扱説明書，EPSON
- EPSON， LP-2000 取扱説明書，EPSON
- Intel，Intel Micro-Processors 1991, Intel
- ボーランドジャパン，Turbo Assembler クイックリファレンスガイト，ボーランドジャパン
- 井上智博，PC-9801/E/F/M グラフィクス解析マニュアル 第三巻，秀和システムトレーディング
- 川村 清，PC-9801 解析マニュアル 第 0 巻，秀和システムトレーディング
- アスキー出版局テクライト編，PC-9801 シリーズ テクニカルデータブック HARDWARE 編，アスキー
- アスキー出版局テクライト編，PC-9801 シリーズ テクニカルデータブック BIOS 編，テクライト編集，アスキー
- 小高輝真・清水和文・速水祐，PC-9801 スーパーテクニック，アスキー
- 速水祐，XMS ドライバの作成，ざべ 1992 年 7 月号，技術評論社
- こうのたけし・小高照真著，PC-9821 の拡張機能解析，ざべ 1993 年 9 月号，技術評論社
- 蒲地輝尚，はじめて読む MASM，アスキー
- 川上峻史，ディスク BIOS と C 言語，工学図書株式会社
- 田辺皓正，マイクロコンピュータシリーズ 15 8086 マイクロコンピュータ，丸善
- 東工大電算機愛好会・小高輝真，98 ハードに強くなる本 2，技術評論社
- 技術評論社編集部，MS-DOS データ活用ハンドブック，技術評論社
- 桜田幸嗣，MS-DOS5 アセンブラプログラミング，アスキー
- 徳田恵一・安部健著，PC-9801/E/F/M インタフェース解析マニュアル，秀和システムトレーディング株式会社

INDEX

■数字

■A

■B

■C

■D

■E

■F

■ス

■セ

■ソ

■タ

■チ

■ツ

■テ

■ヘ

■ホ

■マ

■ミ

■ム

■モ

■ユ

■ヨ

■ラ

■リ

■レ

■ロ

■ワ

■著者紹介　東京理科大学ＥＩＣ（電気工学研究会）

加藤　潔（かとう・きよし）
東京理科大学 物理学科 4 年．'91，'92 年 NEC パソコンアート大賞にて優秀賞受賞．平成 5 年度国家公務員 1 種試験情報工学に合格．画像ローダ"FAR"の作者．

菊地 史陽（きくち・ふみあき）
同 電気工学科 4 年．大学では DES 型の暗号"FEAL"の安全性について研究しています．コンピュータ通信でのハンドルは"SUMIKA"です．見かけたらよろしく．

穴吹　健朗（あなぶき・たけお）
同 数学科 4 年．今回は（も？）お手伝い程度の事しか出来ませんでしたが，それでもサポートしてくださった加藤さん，どうもありがとうございました．

馬木　崇（うまき・たかし）
同 電気工学科 2 年．世間ではマイナーな H98 のユーザーです．最近，As2 をローンでかうという暴挙に出て毎月末にはローンの恐怖に脅えています．

佐藤　淳一（さとう・じゅんいち）
同 経営工学科 4 年．とりあえず第一種情報処理技術者です．RS-232C のことになると黙っていません．98 の本を書いておきながら，今は DOS/V ユーザーです．

菅澤　淳一（すげさわ・じゅんいち）
同 機械工学科 4 年．ソフト会社全部玉砕して，いまはネットのゲームプログラマーとして生活しております．趣味はソフトウェアのインストール　．

貞包　哲男（さだかね・てつお）
同 情報科学科 3 年．EIC の前会長です．この本が発売されるころには，DOS/V ユーザーになる予定です．

佐藤　伸也（さとう・しんや）
同 情報科学科 2 年．ハードソフトを問わず多彩な分野に興味を持ち新入部員を対象とした C 言語講座を開いたりロボット相撲に参加したりとその活動範囲は広い．

PC-9801 プログラマーズ Bible

平成 6 年 3 月 1 日　初　版 第 1 刷 発行
平成 7 年10月 1 日　第 2 版 第 5 刷 発行

著　者　東京理科大学 EIC
発行者　片岡　巌
発行所　株式会社 技術評論社
　東京都新宿区愛住町 8 番地 8
　電話 03（3225）2300 営業部
　　03（3225）3293 編集部
TEX 出力　東京書籍印刷株式会社
印刷／製本 日経印刷株式会社

定価はカバーに表示してあります．

本書の内容に関する御質問は，すべて封書でお願い致します．お電話でのお問い合わせには，一切お答えできません．

ISBN4-87408-615-2 C3055

Printed in Japan

ディスクサービスのお知らせ

本書掲載のサンプルプログラムのソースファイルを，ディスクサービスいたします．御希望の方は，下記の要領で弊社までお申し込みください．

【お送り頂く物】

- 申し込み用紙（コピー可）
  下記の用紙に必要事項を書き込みお送り下さい．
- 100円切手10枚
  送料・ディスク代・手数料込み．

【申込先】

〒160　東京都新宿区愛住町8-8　株式会社技術評論社　第2編集部
PC-9801プログラマーズBible　ディスクサービス　係

・・キリトリ線・・

《PC-9801プログラマーズBibleディスクサービス申込書》

住　所	□□□-□□
氏　名（ふりがな）	
電話番号	
希望メディア	□5.25"(2HD)　□3.5"（2HD）　ともに98フォーマット
PC-9801プログラマーズBibleサービスディスク在中	